

Optimale Generierung und Simulation paralleler Rechensysteme

Dissertation

zur Erlangung des akademischen Grads

eines Doktors der Naturwissenschaften

der Fakultät für Mathematik

der Ludwig-Maximilians Universität, München

vorgelegt von

Dipl.-Math. Michael Grevel

aus Langen

Eingereicht im März 1987



Optimale Generierung und Simulation paralleler Rechensysteme

Dissertation

zur Erlangung des akademischen Grads

eines Doktors der Naturwissenschaften

der Fakultät für Mathematik

der Ludwig-Maximilians Universität, München

vorgelegt von

Dipl.-Math. Michael Grevel

aus Langen

Eingereicht im März 1987

1. Berichterstatter:

Prof. Dr. A. Sachs

2. Berichterstatter:

Prof. Dr. W. Richert

Tag der mündlichen Prüfung: 12. Juni 1987

Inhaltsverzeichnis

	Seite
Vorwort	1
1. Einführung	2
2. Graphentheoretische Grundbegriffe	10
3. Ketten in bewerteten Graphen	14
4. Scheduling	20
5. Lösungssätze von RCS	32
6. Schranken für GRCS	54
7. Anordnungsprobleme	63
8. Lösung von UTAP	71
9. Zuordnungsgraphen	84
10. Lösung von GAP und Verzögerungsoptimierung	100
11. Neue Anfangszeitpunkte	106
12. Modellierung von Algorithmen	111
13. Modellierung paralleler Rechensysteme	131
14. Entwurf und Simulation paralleler Rechenanlagen	141
Schlußfolgerung	148
Literatur	149
Liste der Problemformulierungen	153
Liste der verwendeten Symbole	154
Anhang: Numerisches Beispiel	A1

Vorwort

Der Verfasser möchte an dieser Stelle allen danken, die an der Förderung der vorliegenden Dissertation beteiligt waren:

Herrn Prof. Dr. Sachs für die Formulierung des Themas und die kritische Begleitung der Arbeit, den Herren Dr. Lützeler, Dr. Mrva und Zorn, SIEMENS AG, ZTI SOF 3 für ihre fachkundige Beratung auf verschiedenen Gebieten der Mikroelektronik und "last not least" der SIEMENS AG für die Bereitstellung eines Promotionsstipendiums.

1. Einführung

In den letzten Jahren hat sich immer deutlicher herauskristallisiert, daß Methoden der angewandten Mathematik eine zentrale Rolle beim effizienten Entwurf von Systemen der Mikroelektronik spielen. Dementsprechend wurden mathematische Methoden und abgeleitete Algorithmen (CAD-Systeme) entwickelt, um physikalische und logische Probleme des Chipentwurfs effizient zu lösen /SUDO-83/.

Eines der letzten Entwurfsprobleme der Mikroelektronik, bei dem mathematische Entwicklungssysteme noch kaum eingesetzt werden, ist der Bereich des Entwurfs synchroner Systeme (Mehrprozessorarchitekturen). Dies ist umso gravierender, als durch die Höchstintegration in der VLSI-Technologie im Architekturbereich bisher ungeahnte Möglichkeiten eröffnet werden. Dies schließt insbesondere die Entwicklung von Hochleistungsarchitekturen ein, die auf bestimmte Algorithmen spezialisiert sind (z.B. Pipelines, FFT-Prozessoren, Signalprozessoren).

Hier setzt die vorliegende Arbeit ein:

Es wird ein graphentheoretisches Modell für Algorithmen entwickelt (vergl. /MOLDOVAN-83/, /AHO-79/), um zwei Ziele zu erreichen:

1. Aus einem fest vorgegeben Algorithmus soll eine Architektur eines Parallelrechners generiert werden, auf der dieser Algorithmus besonders effizient ausgeführt werden kann.
2. Auf einer fest vorgegebenen Architektur eines Parallelrechners soll ein Algorithmus unter effizienter Ausnutzung der Fähigkeiten der parallelen Maschine ausgeführt werden.

Diese Algorithmen können in Form eines Programms in einer höheren Programmiersprache, eines Assemblerprogramms nach der Übersetzung durch einen Compiler, oder eines Maschinen- bzw Mikroprogramms gegeben sein.

Das Maschinenmodell beschreibt parallele Rechensysteme (Mehrprozessorarchitekturen), die mit einem gemeinsamen Takt versorgt sind (synchrone Systeme) /LEISERSON-83/, /UHR-84/. Dieser Modellraum umfaßt alle Verschaltungsmöglichkeiten vorgegebener Verarbeitungselemente (Prozessoren). Diese müssen nicht als fertige Bauelemente bzw. Hardwarezellen gegeben sein. Es genügt auch ein Modell, welches die für die funktionale Beschreibungsebene relevanten Eigenschaften der Verarbeitungselemente (Prozessoren) beschreibt.

Parallele Rechensysteme sind beispielsweise MIMD - Architekturen (multiple instructions multiple data) /UHR-84/, SIMD - Rechner (single instructions multiple data) /SIEGEL-85/, systolische Systeme /KUNG-80,81/, MOLDOVAN-83/, /LI-85/, /MELHEM-84/ und Datenpfade von horizontal mikroprogrammierbaren Prozessoren /MEAD-80/. Eine MIMD-Architektur ist ein System (unterschiedlicher) Prozessoren, die untereinander vernetzt sind. Die einzelnen Prozessoren können zu jedem Zeitpunkt unterschiedliche Befehle ausführen. Ein Steuerwerk sammelt die Sende- und Empfangswünsche der einzelnen Prozessoren und stellt Verbindungen bereit. SIMD-Architekturen sind Systeme identischer Prozessoren, die von einem gemeinsamen Steuerwerk gesteuert werden. Zu jedem Zeitpunkt führen die aktivierten Prozessoren den selben Befehl auf unterschiedlichen Daten aus. Der Zeitbedarf für die Befehlsausführung ist für alle Prozessoren gleich. Nach Beendigung der Befehlsausführung werden in einem Takt alle Datentransfers durchgeführt. Auch systolische Systeme bestehen aus gleichartige Prozessoren, die regelmäßig verschaltet sind. Hier entfällt jedoch das Steuerwerk. Die Verarbeitungselemente (Prozessoren) führen bei systolischen Systemen in jedem Systemtakt den gleichen (sehr einfachen) Algorithmus aus. Der gemeinsame Takt ist der einzige globale Steuerungsmechanismus systolischer Systeme. Der Datenpfad eines Mikroprozessors besteht aus ALU (arithmetischer logischer Einheit), Addierer, Shifter usw. (auch in mehrfacher Ausführung), die durch physikalische Datenleitungen untereinander vernetzt sind.

Zur mathematischen Modellierung von Algorithmen verwenden wir den gerichteten azyklischen Graphen. Dieses Modell beschreibt die kausalen Zusammenhänge zwischen Elementaroperationen eines Algorithmus. Diese Operationen sind Programmstücke, Einzelanweisungen, Maschinen- oder Microbefehle.

Ansätze für den automatischen Entwurf von Mehrprozessorarchitekturen als Hardwareimplementierung von speziellen Algorithmen entstammen den Arbeiten von /IBARRA-86/, /JOHNSON-81/, /MOLDOVAN-83,86/, /QUINTON-84/, /RAMAKRISHN-84/, /WEISER-81/. Dort werden Entwurfsmethoden dargestellt, die für regelmäßige Kausalstrukturen systolische Systeme ableiten bzw. auf vorgegebenen systolischen Systemen eine Implementierung versuchen. Dabei wird jedoch nur eine Architektur erzeugt bzw. eine Implementierung angegeben, ohne eine Optimierung vorzunehmen. In /LI-85/ werden aus Algorithmen, die linearen Rekursionen entsprechen, optimale systolische Systeme erzeugt. In der Arbeit von /LEISERSON-83/ werden vorgegebene synchrone Systeme optimiert. Dieser Ansatz wird in der vorliegenden Arbeit weiterverfolgt. Die Codeoptimierung von /HENNESSY-83/, /ARYA-85/ arbeiten mit einer ähnlichen Modellierung von Algorithmen.

men. Dort werden ganzzahlige Optimierungsprobleme formuliert, um Befehlsfolgen für Vektorrechner optimal zu ordnen.

In dieser Arbeit wird nun folgendes Problem gelöst:

Gegeben ist eine Kausalstruktur auf einer vorgegeben Menge von Elementaroperationen. Jede dieser Elementaroperationen benötigt für ihre Implementierung eine Anzahl Ressourcen. Wichtige Typen von Ressourcen sind: Funktionale Elemente (z.B. Addierer, Shifter usw.), Busanschlüsse, Implementierungsfläche, Verlustleistung auf dem Chip, usw. Gegeben sind weiterhin fest vorgegebene synchrone Verarbeitungselemente (Prozessoren), die ihrerseits wiederum verschiedene Ressourcen bereitstellen. Diese Ressourcen sind die für die funktionale Beschreibungsebene relevanten Eigenschaften der Verarbeitungselemente. Für jede dieser Ressourcen ist eine maximale Verfügbarkeit vorgegeben. Weiters kann eine Verbindungstopologie (Verdrahtung) zwischen den Prozessoren gegeben sein.

Gesucht wird für jede Elementaroperation ein Zeitpunkt, an dem die Operation ihre (nach Beginn nicht mehr unterbrechbare) Arbeit beginnt, sowie ein Ort (in Form eines spezifischen Prozessors), auf dem sie implementiert wird.

Diese Zeitpunkte und Orte werden so gewählt, daß eine minimale Gesamtverarbeitungszeit resultiert, wobei gewisse Restriktionen zu beachten sind:

- Die Kausalstruktur muß eingehalten werden.
- Jeder Prozessor, der eine Elementaroperation implementiert, muß die von ihr geforderten Ressourcen bereitstellen.
- Falls eine Operation Daten für eine andere bereitstellt und die beiden Operationen auf verschiedenen Prozessoren implementiert werden, so muß eine Verbindung (Leitung) zwischen den Prozessoren bestehen.
- Die insgesamt zur Verfügung gestellten Ressourcen dürfen nicht überschritten werden.

Falls ein Verbindungsschema der Prozessoren gegeben ist, stellt die Problemlösung eine Simulation der Kausalstruktur (bzw des Programms) auf der durch die Verschaltung vorgegebenen (parallelen) Maschine dar.

Parallelen dieser Arbeit bestehen auch zum "Silicon-Compiling" /SOUTHARD-83/, /AYRES-83/. Beim Silicon-Compiling wird versucht, einen (meist sehr einfachen) Algorithmus oder ein in einer Hardwarebeschreibungssprache dargestelltes System, direkt auf den Werkstoff Silizium zu bringen. Neben den logistischen Fragestellungen (Gegenstand dieser Arbeit) sind auch die geometrischen und

physikalischen Probleme von enormer Bedeutung, da als Ergebnis des Silicon-Compilings Maskenfertigungsunterlagen für die Produktion von Chips erwartet werden. Die hier beschriebene Methode entspricht einer stark formalisierten Sichtweise. Falls die Verarbeitungselemente Modelle von Hardwarezellen sind, kann aus einer Lösung des hier betrachteten Problems ein "floorplan" /WESTE-85/ für eine mögliche Implementierung in Silizium durch eine Platzierung und Verdrahtung erzielt werden.

Die oben dargestellte Problemformulierung führt auf ein ganzzahliges nichtlineares Optimierungsproblem. Dieses Problem wird mit einem zweigeteilten Ansatz gelöst. Die Partitionierung des Gesamtproblems in ein Scheduling- und ein Anordnungsproblem wurde (im einfachen Fall systolischer Systeme) von /QUINTON-84/ vorgeschlagen. Dort werden aber nur systolische Systeme systematisch generiert, ohne eine Optimierung vorzunehmen. Die Arbeit von /ZEMAN-83,84/ zeigt, wie man klassische Scheduling Methoden beim Entwurf von Signalprozessoren benutzt. In dieser Arbeit steht der Anwendungsaspekt (d.h. die Interpretation der Lösung einer Schedulingaufgabe) im Vordergrund. Schedulingprobleme sind schon in den 60er Jahren formuliert worden /HU-61/. Eine weitgehende Übersicht zu diesem Problemkreis bringt /COFFMAN-76/. Diese Problemstellung wurde beim Entwurf von Betriebssystemen für Parallelrechner bzw. Rechnernetze betrachtet. Schon früh /ULLMAN-75,76/, /GAREY-75/ wurde bewiesen, daß die Lösung des "job shop scheduling" allgemein mindestens von der Komplexitätsstufe NP ist. In einigen wenigen Spezialfällen ergaben sich effiziente Algorithmen /COFFMAN-72,76/ und /PAPADIM-79/. Diese Spezialfälle und approximative (heuristische) Algorithmen /COFFMAN-76/, /HU-61/, /KASAHARA-84/ standen im Mittelpunkt der Untersuchungen, da Betriebssysteme für Parallelrechner dieses Problem in Echtzeit lösen müssen.

In dieser Arbeit wird nun ein ressourcenbeschränktes Schedulingproblem formuliert. Die (approximative) Lösung erfolgt mit einem "branch and bound" Verfahren.

Daran anschließend wird ein Anordnungsproblem formuliert, welches in seiner allgemeinsten Form wiederum nur mit einem branch and bound Verfahren gelöst werden kann. Daran anschließend wird ein "retiming" nach /LEISERSON-83/ durchgeführt.



Die vorliegende Arbeit gliedert sich nun folgendermaßen:

In Kapitel 2 werden grundlegende Begriffe der Graphentheorie eingeführt. Die Darstellung folgt dabei weitgehend den Büchern /FISHBURN-85/ und /GOLUMBIC-80/. Abweichend von der üblichen Bezeichnungsweise werden Ketten dargestellt. Eine Kette ist ein geordnetes Paar (K, f) wobei K eine Teilmenge der Knotenmenge und f eine Numerierung dieser Teilmenge ist.

Das dritte Kapitel beweist elementare Eigenschaften von Ketten in pfeilbewerten gerichteten Graphen.

Kapitel 4 formuliert nun neuartige Schedulingaufgaben. Es wird die allgemeine Planungsaufgabe RCS und die Planungsaufgabe für Einheitsoperationen auf identischen Prozessoren ohne weitere Ressourcen UTS formuliert. Höhepunkt dieses Kapitels ist ein Existenzsatz für RCS.

Kapitel 5 beweist Lösungssätze für RCS. Zuerst wird RCS noch einmal verallgemeinert zu GRCS. Dies geschieht, um Teillösungen einfach verknüpfen zu können. Es folgt ein Existenzsatz für GRCS. Falls eine Teilmenge der Quellen der azyklischen Relation zum frühest möglichen Zeitpunkt geplant wird (die Ausführung beginnt), so bleibt auf den restlichen Knoten der Relation ein Teilproblem GRCS' bestehen. Falls eine zulässige (optimale) Lösung von GRCS' gegeben ist, wird gezeigt, wann die zusammengesetzte Lösung zulässig (optimal) ist. Es wird mit Beispielen gezeigt, daß Vereinfachungen von UTS für GRCS nicht mehr (oder nur eingeschränkt) gelten. Diese eingeschränkten Vereinfachungen werden dann bewiesen. Weiters wird bewiesen, daß für die Lösung von GRCS eine Normalform existiert.

Kapitel 6 beweist dann Schranken für GRCS, die für UTS schon seit langem bekannt sind.

Kapitel 7 formuliert Anordnungsprobleme. Zuerst wird das allgemeine Anordnungsproblem GAP definiert. Eine Nebenbedingung von GAP ist jedoch nichtlinear. Um lineare ganzzahlige Probleme zu erhalten, wird diese Nebenbedingung vorerst weggelassen. Hierdurch entsteht die Problemformulierung LAP, bzw. LAP_{con} (für den kontinuierlichen Fall). Als weiter vereinfachte Problemvariante wird das Anordnungsproblem für Einheitsoperationen UTAP definiert.

Die algorithmische Lösung von UTAP wird in Kapitel 8 entwickelt. Es wird ein Existenzsatz für UTAP bewiesen, der die Existenz einer Optimallösung charakterisiert. Höhepunkt von Kapitel 8 ist der Beweis, daß UTAP mit einem Netzflußan-

satz gelöst werden kann. Damit wird außerdem gezeigt, daß die Matrix der Nebenbedingungen von UTAP auf eine total unimodulare Form gebracht werden kann.

Kapitel 9 definiert und untersucht Zuordnungsgraphen. Diese dienen zur algorithmischen Lösung von LAP. Es folgt der Beweis, daß eine optimale Lösung von LAP genau dann existiert, wenn die Stabilitätszahl des Zuordnungsgraphen gleich der Anzahl der Elementaroperationen ist. Nun wird ein Problem CLP definiert, mit dessen Hilfe $(LAP)_{con}$ gelöst werden kann. Anschließend folgt der Beweis, daß die Basislösungen des auf Gleichheitsrestriktionen gebrachten CLP immer dann ganzzahlig sind, wenn der Zuordnungsgraph die Eigenschaft der Perfektheit hat. Diese Eigenschaft des Zuordnungsgraphen ist jedoch nicht notwendig. Dies zeigt ein Gegenbeispiel.

Das Kapitel 10 wendet sich der Lösung von GAP zu. Die algorithmische Grundlage ist der hervorragende Algorithmus LTMIS in /LOUKAKIS-81/. Es wird eine weitere Abbruchregel für LTMIS bewiesen. Weiters wird auf sogenannte "retiming" Strategien nach /LEISERSON-81/ eingegangen.

Das Kapitel 11 schließt den Kreis des Lösungsmöglichkeiten für GAP und RCS. Es wird gezeigt, daß es nicht zwingend notwendig ist, zuerst die Zeit (RCS) und daran anschließend den Ort (GAP) der Ausführung einer Elementaroperation zu bestimmen. Es wird ein einfaches Anordnungsproblem SAP formuliert mit dessen (approximativer) Lösung wiederum ein RCS-artiges Planungsproblem definiert wird. Die Lösungsmethoden sind identisch mit denen für GAP und RCS.

Kapitel 12 wendet sich der Modellierung von Algorithmen zu. Es wird eine Programmiersprache L definiert, die einem eingeschränkten PASCAL entspricht. Daraufhin wird eine Normalform für Algorithmen in L definiert. Der Begriff der kausalen Abhängigkeit von Elementaroperationen wird eingeführt. Der Höhepunkt dieses Kapitels ist der Prototypsatz 17. Hier wird aus einer Lösung eines Prototyp-RCS eine Lösung des zugehörigen RCS konstruiert und die Zulässigkeit bewiesen.

In Kapitel 13 werden parallele Rechenanlagen modelliert. Diese Modelle entsprechen Beschreibungen (auf der RT- bzw Modulebene) folgender Typen von Parallelrechnern:

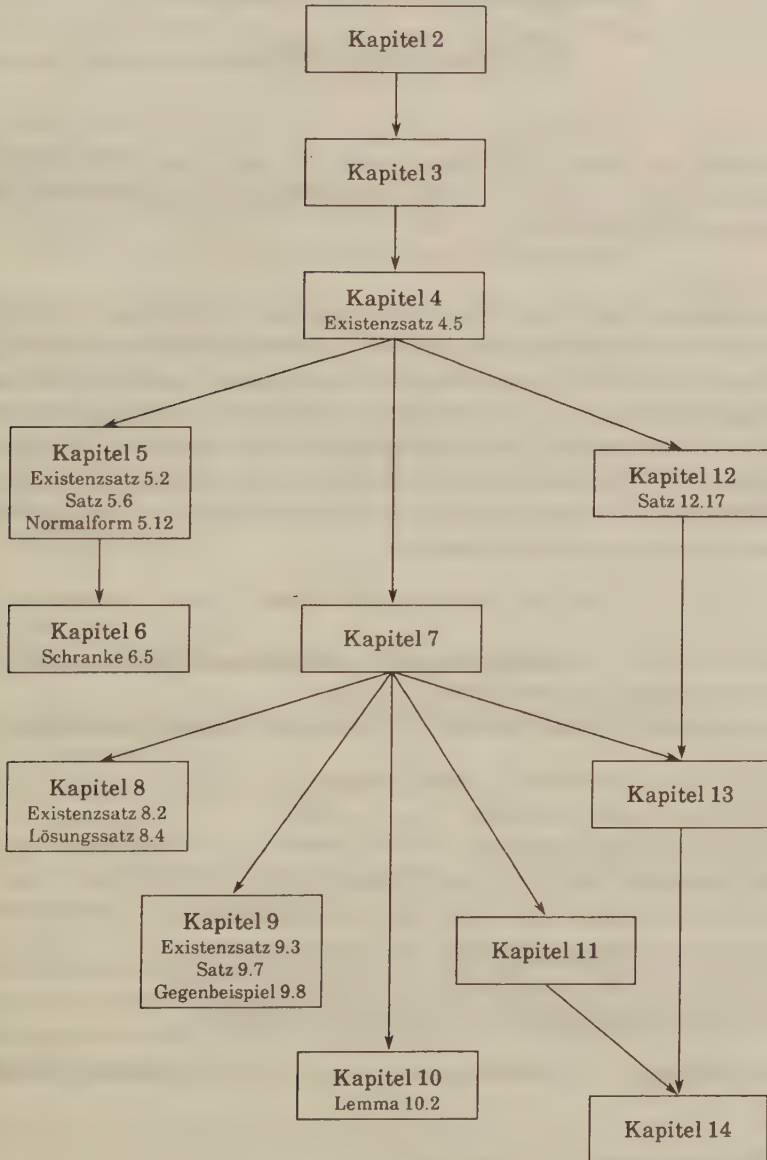
- MIMD-Rechner
- SIMD-Rechner

- Systolische Architekturen
- Datenpfade von horizontal mikroprogrammierbaren Prozessoren

Weiters wird ein Prototyp GAP definiert, aus dessen Lösung eine zulässige Lösung von GAP erzeugt werden kann.

Das Kapitel 14 beschreibt die Einzelschritte zum Entwurf und zur Simulation paralleler Rechenanlagen. Der Rechnertyp bestimmt ganz wesentlich die Parameter von RCS und GAP. Aus den Lösungen von RCS und GAP werden Modelle von Parallelrechnern erzeugt. Falls keine Lösung von RCS bzw. GAP existiert, werden Änderungen der Problemstellung angegeben.

Abhängigkeiten der einzelnen Kapitel und wichtige Ergebnisse:



2. Graphentheoretische Grundbegriffe

In diesem Kapitel werden grundlegende Begriffe über binäre Relationen und (un-)gerichtete Graphen eingeführt. Die Bezeichnungen folgen weitgehend /GOLUM-BIC-80/ und /FISHBURN-85/.

Schreibweisen:

Alle wesentlichen Teile (Definitionen, Sätze, Lemmata,...) eines Kapitels werden durchnummeriert. Querverweise innerhalb eines Kapitels werden nur mit der entsprechenden Nummer angegeben. Verweise über Kapitelgrenzen hinaus werden mit (Kap#.Teil#) bezeichnet. Falls das Ende einer Definition nicht offensichtlich ist (weil z.B. eine unnummerierte Erklärung folgt), wird es mit dem Kürzel "enddef" abgeschlossen. Für Beweise gilt analog das Kürzel "qed". Kommentare in Ableitungsketten werden mit (* text *) gekennzeichnet und dienen der Erklärung. Sie gehören jedoch nicht zur Ableitungskette.

Es ist $N := \{1,2,3,4,\dots\}$ die Menge der strikt positiven ganzen Zahlen. Falls die 0 dazugenommen wird, schreibt man $N_0 := N \cup \{0\}$.

Es sei X eine beliebige, endliche Menge. Es bezeichne $|X|$ die Elementzahl von X .

Es sei $R \subset X \times X$ eine zweistellige (binäre) Relation.

R heißt *reflexiv* : $\Leftrightarrow \Delta X \subset R$, mit $\Delta X := \{(x,x) : x \in X\}$.

R heißt *irreflexiv* : $\Leftrightarrow \Delta X \cap R = \emptyset$.

R heißt *symmetrisch* : $\Leftrightarrow \forall x,y \in X : (x,y) \in R \Rightarrow (y,x) \in R$.

R heißt *antisymmetrisch* : $\Leftrightarrow \forall x,y \in X : (x,y) \in R \Rightarrow (y,x) \notin R$.

R heißt *transitiv* : $\Leftrightarrow \forall x,y,z \in X : (x,y) \in R \wedge (y,z) \in R \Rightarrow (x,z) \in R$.

R heißt *vollständig* : $\Leftrightarrow \forall x,y \in X$ mit $x \neq y : \{(x,y),(y,x)\} \cap R \neq \emptyset$.

Alle in diesem Bericht betrachteten Relationen sind irreflexiv.

Für eine Teilmenge $A \subset X$ bezeichnet $v(A) := A \times A \setminus \Delta A$ die *vollständige Hülle* von A .

$\text{sym}(R) := \{(x,y),(y,x) \in X \times X : (x,y) \in R\}$

nennt man die *symmetrische Hülle* von R .

$c(R) := \{(x,y) \in X \times X \setminus \Delta X : (x,y) \notin \text{sym}(R)\}$ nennt man das *Komplement* von R .

$d(R) := \{(x,y) \in X \times X \setminus \Delta X : (y,x) \in R\}$ nennt man das *Dual* von R . Im Dual werden also alle Paare von R "umgedreht".

Seien R und S zwei Relationen auf X . Die *Komposition* RS von R und S wird wie folgt definiert:

$$RS := \{(x,y) \in X \times X : \exists z \in X \text{ mit } (x,z) \in R \text{ und } (z,y) \in S\}.$$

Insbesondere bezeichnet R^k für $k \in \mathbb{N}$ die k -fache Komposition von R .

$t(R) := R \cup R^2 \cup R^3 \cup \dots$ nennt man die *transitive Hülle* von R .

Das Paar (X,R) heißt der *gerichtete Graph* der Relation R über der *Knotenmenge* X . R heißt auch oft die *Pfeilmenge* des Graphen. Falls die Relation R symmetrisch ist, existiert für jeden Pfeil $(x,y) \in R$ auch der "Gegenpfeil" $(y,x) \in R$. Da in diesem Fall die Richtung der Verbindung zweier Knoten keine Rolle spielt, heißt das Paar (X,R) der *ungerichtete Graph* der (symmetrischen) Relation R über der Knotenmenge X . R heißt dann auch die *Kantenmenge* zum Graphen (X,R) . Eine Kante aus R wird ebenfalls in Paarschreibweise mit (x,y) bezeichnet. Sie identifiziert die zwei Pfeile (x,y) und (y,x) .

Ein Element $x^* \in X$ heißt *minimales Element* oder *Quelle* von R , falls gilt:

$$\forall y \in X \setminus \{x^*\} : (y,x^*) \notin R$$

Die Menge aller Quellen in (X,R) wird mit Q_R bezeichnet.

Ein Element $z^* \in X$ heißt *maximales Element* oder *Senke* von R , falls gilt:

$$\forall y \in X \setminus \{z^*\} : (z^*,y) \notin R$$

Die Menge aller Senken wird mit S_R bezeichnet.

Diese Definitionen sind in erster Linie für gerichtete Graphen sinnvoll. Für ungerichtete Graphen sind minimale bzw maximale Elemente *isolierte* (d.h. von keiner Kante getroffene) Knoten.

Falls man sich die Pfeilinterpretation von gerichteten Graphen vergegenwärtigt, ergeben sich ganz natürlich *Vorgänger* und *Nachfolger* eines Knotens. Die *Vorgängermenge* $V(x)$ und die *Nachfolgermenge* $N(x)$ eines Knotens $x \in X$ werden wie folgt definiert:

$$V(x) := \{y \in X : (y,x) \in R\}, \quad V^+(x) := V(x) \cup \{x\};$$

$$N(x) := \{y \in X : (x,y) \in R\}, \quad N^+(x) := N(x) \cup \{x\};$$

Für ungerichtete Graphen (X, R) sind für jeden Knoten die Vorgängermengen und die Nachfolgermenge identisch. Deshalb definiert man die *Nachbarschaftsmenge* $A(x)$ eines Knotens $x \in X$ folgendermaßen:

$$A(x) := \{y \in X : (y, x) \in R \text{ oder } (x, y) \in R\}, \quad A^+(x) := A(x) \cup \{x\};$$

Falls für einen Graphen (X, R) nur eine bestimmte Teilrelation $E \subset R$ betrachtet werden soll, heißt (X, E) ein *Teilgraph* zu (X, R) . Dann wird die Vorgängermenge $V_E(x)$ bezüglich E und die Nachfolgermenge $N_E(x)$ bezüglich E eines Knotens $x \in X$ wie folgt definiert:

$$V_E(x) := \{y \in X : (y, x) \in E\}, \quad V_E^+(x) := V_E(x) \cup \{x\};$$

$$N_E(x) := \{y \in X : (x, y) \in E\}, \quad N_E^+(x) := N_E(x) \cup \{x\};$$

Genauso wie man Teilrelationen betrachtet, kann man für eine Teilmenge A von X den von A in $G := (X, R)$ induzierten *Untergraphen* $G(A) := (A, R(A))$ definieren, wobei

$$R(A) := R \cap v(A) \quad \text{heißt die von } A \text{ induzierte Teilrelation.}$$

Zwei wichtige Spezialfälle werden nun betrachtet:

Falls $R(A) = \emptyset$, so heißt A eine *stabile Menge*.

Für ungerichtete Graphen definiert man:

Falls $R(A) = v(A)$, so heißt A eine *vollständige Menge* oder auch *Clique*.

Die *Stabilitätszahl* $\sigma(G)$ wird wie folgt definiert:

$$\sigma(G) := \max \{ |A| : A \subset X \text{ und } A \text{ stabil} \}.$$

Für ungerichtete Graphen wird die *Cliquenzahl* $\omega(G)$ wie folgt definiert:

$$\omega(G) := \max \{ |A| : A \subset X \text{ und } A \text{ vollständig} \}.$$

Es sei (X, R) ein gerichteter Graph. Eine Teilmenge K heißt *Kette* in (X, R) , falls es eine Bijektion $f: \{1, \dots, |K|\} \rightarrow K$ gibt, so daß gilt:

$$\forall i = 1, \dots, |K|-1 : (f(i), f(i+1)) \in R.$$

Falls für eine Kette K die entsprechende Bijektion f genannt werden soll, schreibt man auch (K, f) . Da f eine Bijektion ist, sind per Definition alle Knoten einer Kette paarweise verschieden. Die *Pfeilmenge* $R_{K,f}$ einer Kette (K, f) wird folgendermaßen

definiert:

$$R_{K,f} := \{ (f(j), f(j+1)) : j = 1, \dots, K-1 \}$$

Eine Kette heißt *Zyklus*, falls zusätzlich gilt: $(f(K), f(1)) \in R$

Ein gerichteter Graph heißt *azyklisch*, wenn er keinen Zyklus enthält oder äquivalenterweise:

$$t(R) \cap \Delta X = \emptyset.$$

Ein azyklischer Graph (X, R) heißt genau dann *Ordnung*, wenn seine Relation R transitiv ist oder kurz:

$$t(R) = R.$$

3. Ketten in bewerteten Graphen

In diesem Kapitel werden beliebige irreflexiven Relationen R über einer endlichen Menge X betrachtet. Die Pfeilmenge R wird nun beliebig reell bewertet:

$$c : R \longrightarrow \mathbb{R}$$

Es wird in Lemma 2 bewiesen, daß stets eine (bezüglich der Bewertung) längste Kette in (X, R) existiert. Weiters wird gezeigt, daß die längsten Ketten auch nur über eine Teilrelation R_0 bestimmt werden können.

1. Definition.

Es sei $G := (X, R)$ der gerichtete Graph einer irreflexiven Relation R über X . Es sei $c : R \longrightarrow \mathbb{R}$ eine beliebige Pfeilbewertung von G und es sei (K, f) eine Kette in G . Die Länge $l_c(K, f)$ der Kette (K, f) wird wie folgt definiert:

$$l_c(K, f) := \sum_{j=1}^{/K/-1} c(f(j), f(j+1))$$

Falls (K, f) ein Zyklus in (X, R) ist, wird die Länge (Bewertung) $l_c^*(K, f)$ folgendermaßen definiert:

$$l_c^*(K, f) := l_c(K, f) + c(f(/K/), f(1))$$

Ein Zyklus heißt *negativ (bewertet)*, falls $l_c^*(K, f) < 0$

Es wird nun gezeigt, daß es stets eine längste Kette in (X, R) gibt.

2. Lemma.

Es sei (X, R) ein gerichteter Graph und es sei $c : R \longrightarrow \mathbb{R}$ eine beliebige Pfeilbe-

wertung. Dann existiert eine längste Kette in (X, R) .

Beweis:

Die Menge aller Ketten in (X, R) ist endlich.

Begründung:

Sei $\forall j = 1, \dots, |X|$: $n_j := |\{(K, f) : (K, f) \text{ Kette in } (X, R) \text{ mit } |K| = j\}|$

Jede Kette (K, f) stellt eine Permutation auf der Menge K dar. Somit folgt:

$$\forall j = 1, \dots, |X| : n_j \leq j!$$

Damit gilt insgesamt:

$$|\{(K, f) : (K, f) \text{ Kette in } (X, R)\}| = \sum_{j=1}^{|X|} n_j \leq \sum_{j=1}^{|X|} j! < \infty$$

Da es nur endlich viele Ketten in (X, R) gibt, gibt es stets eine längste Kette. $\quad$ qed.

Ein wichtiger Spezialfall entsteht, falls für die Pfeilbewertung c gilt:

$$\forall x \in X \quad \forall y \in V(x) : c(y, x) = \text{const}$$

In diesem Fall ergibt sich:

3. Lemma.

Es sei (X, R) ein gerichteter Graph und es sei $c : R \longrightarrow \mathbb{R}$ eine beliebige Pfeilbewertung. Sei weiterhin $\tau : X \longrightarrow \mathbb{R}$ eine Knotenbewertung, so daß gilt:

$$\forall x \in X \quad \forall y \in V(x) : c(y, x) = \tau(x)$$

Dann gilt für jede Kette (K, f) in (X, R) :

$$l_c(K, f) = \sum_{j=2}^{|K|} \tau(f(j))$$

Beweis:

Es sei (K, f) eine Kette in (X, R) . Dann gilt:

$$l_c(K, f) = \sum_{j=1}^{/K/-1} c(f(j), f(j+1)) = \sum_{j=1}^{/K/-1} \tau(f(j+1)) = \sum_{j=2}^{/K/} \tau(f(j))$$

qed.

In diesem Fall heißt die längste Kette, die in einem Knoten $x \in X$ beginnt, die *Höhe* des Knotens x unter der Knotenbewertung τ . (vergl. /COFFMAN-76/, /DOLEV-84/, /KASAHARA-84/).

Es wird nun gezeigt, daß die Bestimmung der längsten Ketten in (X, R) über eine Teilrelation R_0 erfolgen kann.

4. Lemma.

Es sei (X, R) der gerichtete Graph einer azyklischen Relation R über X und es sei $c: R \rightarrow \mathbb{R}$ eine beliebige Pfeilbewertung.

$$M := \{ (x, y) \in R : \exists (K, f) \text{ Kette in } (X, R) \text{ mit } f(1) = x \text{ und } f(/K/) = y \text{ und } /K/ > 2 \text{ und } c(x, y) \leq l_c(K, f) \}$$

Dann ist die Länge der längsten Kette in (X, R) und in (X, R_0) mit $R_0 := R \setminus M$ gleich.

Beweis:

Da R_0 eine Teilmenge von R ist, ist die längste Kette in (X, R_0) nie größer als eine längste Kette in (X, R) .

Zeige nun, daß eine längste Kette in (X, R) so gewählt werden kann, daß sie in R_0 liegt.

Aus der Menge der längsten Ketten in (X, R) wähle eine Kette (K, f) aus, sodaß für alle anderen längsten Ketten (K', f') gilt:

$$/K/ \geq /K'/.$$

Ann. $\exists j \in \{1, \dots, /K/-1\} : (f(j), f(j+1)) \notin R_0$.

Damit liegt $(f(j), f(j+1))$ in der Menge M . Nach Definition von M wähle nun eine Kette (K_0, f_0) in (X, R) mit

- (i) $/K_0/ > 2$
- (ii) $f_0(1) = f(j)$ und $f_0(/K_0/) = f(j+1)$
- (iii) $c(f(j), f(j+1)) \leq l_c(K, f)$

Definiere nun eine Kette (K^*, f^*) in (X, R) , die nicht den Pfeil $(f(j), f(j+1))$ enthält, sondern den "Umweg" (K_0, f_0) .

$$K^* := K \cup K_0$$

Behauptung: $K \cap K_0 = \{f(j), f(j+1)\}$

" $\subset$ " ist offensichtlich nach (ii).

" $\supset$ " Ann.: Es gibt ein $z \in K \cap K_0 : j^* := f^{-1}(z) \neq j, j+1$ und $k^* := f_0^{-1}(z) \neq 1, /K_0/$

1. Fall. $j^* < j$.

Entweder ist $k^* = 2$. Dann existiert eine Kette von $f(j^*)$ nach $f(j)$ als Teilkette von (K, f) . Sie ist aber ein Zyklus, da

$$(f(j), f(j^*)) = (f_0(1), z) = (f_0(1), f_0(k^*)) = (f_0(1), f_0(2)) \in R$$

Andernfalls ($k^* > 2$) gibt es eine Kette von $f(j)$ nach $f(j^*) = f_0(1)$ und von da aus weiter nach $f_0(k^*-1)$. Diese Kette ist ebenfalls ein Zyklus, da

$$(f_0(k^*-1), f(j)) = (f_0(k^*-1), z) = (f_0(k^*-1), f_0(k^*)) \in R$$

In beiden Fällen ergibt sich ein Widerspruch zur Azyklität von (X, R) .

2. Fall. $j^* > j+1$.

Entweder ist $k^* = /K_0/-1$. Dann gibt es eine Kette von $f_0(/K_0/) = f(j+1)$ nach $z = f(j^*)$ als Teilkette von (K, f) . Diese Kette ist jedoch ein Zyklus, da

$$(z, f_0(/K_0/)) = (f_0(k^*), f_0(/K_0/)) = (f_0(/K_0/-1), f_0(/K_0/)) \in R$$

Andernfalls ($k^* < /K_0/-1$) gibt es eine Kette von $f_0(k^*+1)$ nach $f(j+1)$ und von da aus weiter nach $z = f_0(k^*)$. Diese Kette ist jedoch ebenfalls ein Zyklus.

Damit wurde gezeigt, daß $K \cap K_0 = \{f(j), f(j+1)\}$ woraus $/K^*/ = /K/ + /K_0/-2$ folgt.

Definiere nun eine Abbildung $f^* : \{1, \dots, /K/ + /K_0/-2\} \longrightarrow K^*$

$$f^*(l) := \begin{cases} f(l) & \text{für } l = 1, \dots, j \\ f_0(l+1-j) & \text{für } l = j+1, \dots, j + /K_0/-2 \\ f(l+2 - /K_0/) & \text{für } l = j + /K_0/-1, \dots, /K/ + /K_0/-2 \end{cases}$$

Da $f^*({1, \dots, j, j + /K_0/-1, \dots, /K/ + /K_0/-2}) = K$ und

$f^*({j+1, j + /K_0/-2}) = K_0 \setminus \{f_0(1), f_0(/K_0/)\}$ ist f^* injektiv. Da der Definitionsbereich von f^* endlich ist, ist f^* damit auch bijektiv.

Nach Definition von f^* gilt: $\forall i \in \{1, \dots, /K/ + /K_0/-3\} : (f^*(i), f^*(i+1)) \in R$

Da (K, f) eine längste Kette in (X, R) ist, gilt:

$$\begin{aligned} l_c(K, f) \geq l_c(K^*, f^*) &= \sum_{i=1}^{/K^*/-1} c(f^*(i), f^*(i+1)) = \\ &= \sum_{i=1}^{j-1} c(f^*(i), f^*(i+1)) + c(f^*(j), f^*(j+1)) + \sum_{i=j+1}^{j + /K_0/-3} c(f^*(i), f^*(i+1)) + \\ &+ c(f^*(j + /K_0/-2), f^*(j + /K_0/-1)) + \sum_{i=j + /K_0/-1}^{/K/ + /K_0/-3} c(f^*(i), f^*(i+1)) = \\ &= \sum_{i=1}^{j-1} c(f(i), f(i+1)) + c(f_0(1), f_0(2)) + \sum_{i=j+1}^{j + /K_0/-3} c(f_0(i+1-j), f_0(i+2-j)) + \\ &+ c(f_0(/K_0/-1), f_0(/K_0/)) + \sum_{i=j + /K_0/-1}^{/K/ + /K_0/-3} c(f(i+2 - /K_0/), f^*(i+3 - /K_0/)) = \\ &= \sum_{k=1}^{j-1} c(f(k), f(k+1)) + l_c(K_0, f_0) + \sum_{k=j+1}^{/K/-1} c(f(k), f(k+1)) \geq l_c(K, f) \end{aligned}$$

Die letzte Ungleichung gilt mit (ii) und (iii). Damit ist (K^*, f^*) auch eine längste Kette in (X, R) , für die gilt:

$$/K^*/ = /K/ + /K_0/ - 2 > /K/ + 2 - 2 = /K/.$$

Dies ist ein Widerspruch zur Wahl von (K, f) .

Somit liegen alle Pfeile von (K, f) in (X, R_0) .

qed.

4. Scheduling

In diesem Kapitel wird eine verallgemeinerte Scheduling- bzw Planungsaufgabe formuliert. Die Verallgemeinerung ergibt sich durch die Einführung von minimalen und maximalen Zeitabständen. Der Höhepunkt dieses Kapitels ist ein Existenzsatz für die verallgemeinerte Planungsaufgabe.

Die erste Arbeit über Planungsaufgaben datiert aus dem Jahr 1961 /HU-61/. Einen guten Überblick über Formulierungs- und Lösungsmöglichkeiten bietet /COFFMAN-76/. Früh zeichnete sich schon ab, daß allgemeine Planungsaufgaben nur durch vollständige Enumeration gelöst werden können /ULLMAN-75/. Daher konzentrierte sich die Forschungsarbeit auf effizient lösbare Spezialfälle, auf Komplexitätsbetrachtungen und auf Heuristiken zur Lösung allgemeiner Planungsprobleme. In diesem Kapitel werden sog. "job-shop" Probleme formuliert. Ein *Job* oder auch eine *Elementaroperation* ist eine unteilbare (nonpreemptive) Arbeitseinheit. Falls diese Arbeitseinheit begonnen wird, kann sie bis zu ihrem Ausführungsende nicht mehr unterbrochen werden. Die Zeitspanne der Ausführung einer Elementaroperation bestimmt ihr *Ausführungsintervall*. Damit eine Elementaroperation ausgeführt werden kann, müssen gewisse Vorbedingungen erfüllt sein. Diese sind:

1. Die Vollendung anderer Elementaroperationen.

2. Die Verfügbarkeit von Ressourcen.

Ressourcen sind Betriebsmittel, die einer Elementaroperation zur Verfügung gestellt werden müssen, damit sie ausgeführt werden kann. Während des Ausführungsintervalls sind die Ressourcen, die eine Elementaroperation angefordert hat, belegt und können damit nicht von anderen Elementaroperationen benutzt werden. Die Ressourcen zerfallen in verschiedene Ressourcenklassen, wobei die einzelnen Klassen jeweils eine vorgegebene Anzahl von Objekten enthalten. Ziel einer Planungsaufgabe ist es, die Gesamtausführungszeit aller Elementaroperationen möglichst klein zu machen.

Die Voraussetzung (1) kann einfach in Graphenschreibweise übertragen werden. Auf der Menge der Elementaroperationen besteht ein Pfeil von Job x nach Job y genau dann, wenn Operation y erst dann ausgeführt werden darf, wenn Operation x beendet ist.

3. Bezeichnungen.

Es sei T die endliche Menge der Elementaroperationen. Die Relation $R \subset T \times T$ beschreibe das zeitliche Nacheinander der Elemente von T gemäß (1). Falls die Relation R einen Zyklus enthält, müssen die Elementaroperationen dieses Zyklus jeweils zeitlich nacheinander ausgeführt werden. In diesem Fall ist die Planungsaufgabe stets unlösbar. Die Knotenbewertung $\tau: X \longrightarrow N$ ($t: X \longrightarrow N$) gibt für jede Elementaroperation die Länge des Ausführungsintervalls (den Bearbeitungsbeginn) an. Für jede Elementaroperation $x \in T$ ergibt sich das Bearbeitungsende aus $t(x) + \tau(x)$. Für jedes Element (x, y) der azyklischen Relation R sei $\lambda(x, y)$ bzw. $\kappa(x, y)$ der minimale bzw. maximale zeitliche Abstand zwischen $t(x)$ und $t(y)$ an; d.h. $\lambda, \kappa: R \longrightarrow N$. Ein Nachfolger y von x darf seine Ausführung erst beginnen, wenn x beendet ist. Da der früheste Ausführungsbeginn $t(x) + \lambda(x, y)$ von y erst zeitlich hinter dem Ausführungsende $t(x) + \tau(x)$ von x liegen muß, wird folgende Bedingung stets gefordert:

$$\forall (x, y) \in R: \quad \tau(x) \leq \lambda(x, y)$$

Der Wert $t_{\max}$ gibt eine obere Schranke für den Ausführungsbeginn aller Elementaroperationen an. Es sei $\Phi := \{\phi_1, \dots, \phi_k\}$ die Menge der verschiedenen Ressourcen. Die Abbildung $\mu: T \longrightarrow N_0^k$ gibt die Ressourcenanforderung der Elementaroperationen an. Für $x \in T$ gibt $\mu(x)_i$ die Menge der Resource ϕ_i an, die Job x benötigt. Es sei $s \in N_0^k$ der Vektor der Gesamtverfügbarkeiten der Ressourcen.

4. Definition. (RCS - "resource constrained scheduling")

Gegeben seien $T, R, \tau, \Phi, \mu, s, \lambda, \kappa$ und $t_{\max}$ aus (3)., wobei R eine azyklische Relation über T ist. Gesucht ist eine Abbildung $t: T \longrightarrow N$ mit

$$\max \{t(x) + \tau(x) : x \in T\} \longrightarrow \min$$

unter folgenden Nebenbedingungen:

$$(a) \quad \forall (x, y) \in R: \quad t(x) + \lambda(x, y) \leq t(y)$$

$$(b) \quad \forall (x, y) \in R: \quad t(x) + \kappa(x, y) \geq t(y)$$

- (c) $\forall \alpha = 1, \dots, t_{\max} :$ $T_\alpha := \{x \in T : t(x) \leq \alpha < t(x) + \tau(x)\}$
 $\forall \alpha = 1, \dots, t_{\max} :$ $\sum_{x \in T_\alpha} \mu(x) \leq s$ (komponentenweise)
- (d) $\forall x \in T :$ $t(x) \leq t_{\max}$

Das Ausführungsintervall einer Elementaroperation ist $[t(x), t(x) + \tau(x)[$. Der Zielfunktionswert $\max \{t(x) + \tau(x) : x \in T\}$ gibt den Zeitpunkt an, an dem alle Elementaroperationen beendet sind. Eine Optimallösung minimiert den Zielfunktionswert unter den Nebenbedingungen (a), (b), (c) und (d). Die Nebenbedingungen (a), (b) beschreiben das zeitliche Nacheinander der Elementaroperationen $x, y \in T$ mit $(x, y) \in R$, wobei der Zeitabstand zwischen Anfang von x und Anfang von y im Intervall $[\lambda(x, y), \kappa(x, y)]$ liegt. Für die Nebenbedingung (c) ist T_α die Menge der Elementaroperationen, die zum Zeitpunkt α in Bearbeitung sind. Die Nebenbedingung (c) fordert, daß zu jedem Zeitpunkt α nur so viele Ressourcen benötigt werden, wie insgesamt verfügbar sind. Für eine Elementaroperation x gibt $\tau(x)$ die Belegungsdauer der Ressourcen an, während für alle Nachfolger y von x der frühest mögliche Ausführungsbeginn durch $t(x) + \lambda(x, y)$ festgelegt ist. Eine obere Schranke für den Ausführungsbeginn eines Nachfolgers y von x ist $t(x) + \kappa(x, y)$. Die Nebenbedingung (d) fordert die Einhaltung der oberen Schranke für den Ausführungsbeginn aller Elementaroperationen. Aus RCS entsteht die klassische Formulierung, falls gilt:

$$\forall (x, y) \in R : \quad \lambda(x, y) = \tau(x) \quad \text{und} \quad \kappa(x, y) = \infty$$

Es folgt nun ein Existenzsatz für RCS. Die folgenden drei Bedingungen garantieren die Existenz einer Lösung von RCS.

5. Existenzsatz.

Gegeben sei RCS. Es gelte:

- (i) In $(T, \text{sym}(R))$ mit der Pfeilbewertung $c : \text{sym}(R) \longrightarrow \mathbb{Z}$

$$c(x, y) := \begin{cases} \kappa(x, y) & \text{für } (x, y) \in R \\ -\lambda(y, x) & \text{für } (x, y) \in d(R) \end{cases}$$

gibt es keinen negativ bewerteten Zyklus.

$$(ii) \quad \forall S \subset T \text{ t(R)-stabil: } \sum_{x \in S} \mu(x) \leq s$$

$$(iii) \quad t_{\max} \geq c_{\max} * (T/ - 1) + 1 \quad c_{\max} := \max \{c(x,y) : (x,y) \in \text{sym}(R)\}$$

Dann existiert eine optimale Lösung von RCS.

Beweis:

Da R eine azyklische Relation ist, ist R insbesondere antisymmetrisch. Damit ist $\text{sym}(R)$ die disjunkte Vereinigung von R und $d(R)$. Damit ist die Pfeilbewertung c wohldefiniert.

Im ersten Beweispunkt wird sogar etwas mehr gezeigt, als im Existenzsatz gefordert wird:

(*) Die Bedingung (i) ist **genau** dann erfüllt, wenn es eine Abbildung $t: T \longrightarrow \mathbb{N}$ gibt, sodaß gilt:

$$\forall (x,y) \in R: \quad t(y) - t(x) \leq \kappa(x,y)$$

$$\forall (x,y) \in R: \quad t(x) - t(y) \leq -\lambda(x,y)$$

Zum Beweis der Aussage (*) wird ein Hilfsatz benötigt:

Hilfssatz:

Es sei (X,E) der gerichteter Graph einer irreflexiven Relation E über einer endlichen Menge X mit Pfeilbewertung $c: E \longrightarrow \mathbb{R}$. (X,E) habe keinen negativ bewerteten Zyklus. Es sei $\psi: E \longrightarrow \mathbb{R}$ eine Abbildung, die die Bedingungen (FB) und (NN) erfüllt. Dann erfüllt ψ die Bedingung (ZF)

$$(FB) \quad \forall x \in X: \quad \sum_{y \in N_E(x)} \psi(x,y) - \sum_{y \in V_E(x)} \psi(y,x) = 0$$

$$(NN) \quad \forall (x,y) \in E: \psi(x,y) \geq 0$$

$$(ZF) \quad \sum_{(x,y) \in E} c(x,y) * \psi(x,y) \geq 0$$

Beweis des Hilfssatzes:

Es sei $\psi : E \longrightarrow \mathbb{R}$ eine Abbildung, die (FB) und (NN) erfüllt. Solche Abbildungen werden in der Literatur z.B. /GARFINKEL-72/ Flüsse genannt.

Der Nachweis der Ungleichung (ZF) erfolgt durch vollständige Induktion nach $|E|$. Falls $|E| = 1$, so ist wegen der Irreflexivität von E $\psi \equiv 0$ die einzige zulässige Lösung und somit optimale Lösung.

Sei nun $|E| > 1$:

Falls es ein $(x,y) \in E$ gibt mit $\psi(x,y) = 0$, so betrachte die Problemstellung für $(X, E \setminus \{(x,y)\})$ und wende die Induktionsannahme an.

Es sei nun $\forall (x,y) \in E : \psi(x,y) > 0$. Definiere nun $\varepsilon := \min \{ \psi(x,y) : (x,y) \in E \}$. Sei nun $\varepsilon = \psi(x',y')$. Wegen der Flußbedingung (FB) und der Nichtnegativität von ψ , gibt es einen Zyklus (K,f) in (X,E) mit $f(1) = y'$ und $f(K) = x'$, sodaß gilt:

$$\forall j = 1, \dots, K-1 : \psi(f(j), f(j+1)) \geq \varepsilon.$$

$$E_K := \{ (f(j), f(j+1)) : j = 1, \dots, K-1 \} \cup \{ (x', y') \}$$

Die Menge E_K ist die Pfeilmenge des Zyklus (K,f) . Definiere nun eine Abbildung $\psi' : E \longrightarrow \mathbb{R}$:

$$\psi'(x,y) := \begin{cases} \psi(x,y) - \varepsilon & \text{für } (x,y) \in E_K \\ \psi(x,y) & \text{sonst} \end{cases}$$

Die Nichtnegativitätsbedingung (NN) ist nach der Definition von ε erfüllt. Die Flußbedingung (FB) ist für alle Knoten aus $X \setminus K$ erfüllt, da sich keine Änderungen an ψ ergeben. Falls $z \in K$ so hat z genau einen Vorgänger und genau einen Nachfolger in E_K , sodaß sich die Flußkorrektur bezüglich ε aufhebt.

Nach Induktionsannahme erfüllt nun ψ' die Bedingung (ZF), da ψ' auf (x',y') den Wert 0 hat. Für (ZF) von ψ ergibt sich:

$$\sum_{(u,v) \in E} c(u,v) * \psi(u,v) = \sum_{(u,v) \in E} c(u,v) * \psi'(u,v) + \sum_{(u,v) \in E_K} c(u,v) * \varepsilon =$$

$$\sum_{(u,v) \in E} c(u,v) * \psi'(u,v) + \varepsilon * \sum_{(u,v) \in E_K} c(u,v) \geq \sum_{(u,v) \in E} c(u,v) * \psi'(x,y) \geq 0$$

Die vorletzte Ungleichung gilt wegen $\varepsilon > 0$ und der Nichtnegativität der Bewertungen aller Zyklen in (X, E) .

Dies beendet den Beweis des Hilfssatzes.

Nun folgt der Beweis der Aussage (*):

"immer dann". Es sei $(T, \text{sym}(R))$ mit Pfeilbewertung c ohne negativen Zyklus.

Betrachte nun folgendes lineare Optimierungsproblem (P):

Gegeben ist $c_0 : T \longrightarrow \mathbb{R}$ mit $c_0 \equiv 0$

Gesucht ist eine Abbildung $t : T \longrightarrow \mathbb{R}$, sodaß:

$$c_0 T t := \sum_{x \in T} c_0(x) * t(x) \longrightarrow \max$$

unter den Nebenbedingungen:

Dualvariable

$$\forall (x, y) \in R : \quad t(y) - t(x) \leq \kappa(x, y)$$

$$u(x, y) \geq 0$$

$$\forall (x, y) \in R : \quad t(x) - t(y) \leq -\lambda(x, y)$$

$$v(x, y) \geq 0$$

An die Abbildung t ist keine Vorzeichenbedingung gestellt. Das zu (P) duale Programm (D1) /LUENBERGER-72/ lautet folgendermaßen:

Gesucht sind Abbildungen $u, v : T \longrightarrow \mathbb{R}$, sodaß:

$$\sum_{(x, y) \in R} \kappa(x, y) * u(x, y) - \lambda(x, y) * v(x, y) \longrightarrow \min$$

unter den Nebenbedingungen:

$$\forall x \in T : \quad \sum_{z \in N_R(x)} -u(x, z) + v(x, z) + \sum_{z \in V_R(x)} u(z, x) - v(z, x) = 0$$

$$\forall (x, y) \in R : \quad u(x, y) \geq 0$$

$$\forall (x, y) \in R : \quad v(x, y) \geq 0$$

Damit der Hilfsatz angewendet werden kann, wird noch ein zu (D1) äquivalentes Programm (D2) definiert werden:

Gesucht ist eine Abbildung $\psi : \text{sym}(R) \longrightarrow \mathbb{R}$, sodaß:

$$\sum_{(x, y) \in \text{sym}(R)} c(x, y) * \psi(x, y) \longrightarrow \min \quad \text{mit } c \text{ aus (i)}$$

unter den Nebenbedingungen:

$$\forall x \in T: \sum_{z \in N_{\text{sym}(R)}(x)} \psi(x,z) - \sum_{z \in V_{\text{sym}(R)}(x)} \psi(z,x) = 0$$

$$\forall (x,y) \in R: \psi(x,y) \geq 0$$

Das Problem (D2) zusammen mit der Zyklusbedingung (i) erfüllt die Voraussetzungen des Hilfsatzes. Damit haben alle für (D2) zulässigen Lösungen nichtnegativen Zielfunktionswert. Da $\psi \equiv 0$ zulässig ist, ist die Nullabbildung auch optimal für (D2).

Zeige nun: Für alle u, v zulässig für (D1) gibt es ein ψ zulässig für (D2), sodaß:

$$\kappa T u - \lambda T v = c T \psi.$$

Es sei nun u, v zulässig für (D1). Definiere eine Abbildung $\psi: \text{sym}(R) \rightarrow \mathbb{R}$:

$$\psi(x,y) := \begin{cases} u(x,y) & \text{für } (x,y) \in R \\ v(y,x) & \text{für } (x,y) \in d(R) \end{cases}$$

Behauptung: Die Abbildung ψ ist zulässig für (D2).

Begründung: Nach Definition von $\text{sym}(R)$ und mit R antisymmetrisch gilt:

$$N_{\text{sym}(R)}(x) = N_R(x) \cup V_R(x) = V_{\text{sym}(R)}(x) \quad (\text{disjunkte Vereinigung}).$$

Sei $x \in T$:

$$\begin{aligned} \sum_{z \in N_{\text{sym}(R)}(x)} \psi(x,z) - \sum_{z \in V_{\text{sym}(R)}(x)} \psi(z,x) &= \sum_{z \in N_R(x)} \psi(x,z) + \sum_{z \in V_R(x)} \psi(x,z) - \sum_{z \in V_R(x)} \psi(z,x) - \sum_{z \in N_R(x)} \psi(z,x) = \\ \sum_{z \in N_R(x)} u(x,z) + \sum_{z \in V_R(x)} v(z,x) - \sum_{z \in V_R(x)} u(z,x) - \sum_{z \in N_R(x)} v(x,z) &= 0 \end{aligned}$$

Damit ist ψ zulässig für (D2) und es gilt:

$$\kappa T u - \lambda T v = \sum_{(x,y) \in R} \kappa(x,y) * u(x,y) - \lambda(x,y) * v(x,y) =$$

$$\sum_{(x,y) \in R} \kappa(x,y) * u(x,y) + \sum_{(x,y) \in R} -\lambda(x,y) * v(x,y) =$$

$$\sum_{(x,y) \in R} c(x,y) * \psi(x,y) + \sum_{(x,y) \in R} c(y,x) * \psi(y,x) =$$

$$\sum_{(x,y) \in R} c(x,y) * \psi(x,y) + \sum_{(x,y) \in d(R)} c(x,y) * \psi(x,y) = c^T \psi$$

Somit gilt für jede zulässige Lösung u, v für (D1) $\kappa^T u - \lambda^T v \geq 0$. Da die Nullfunktionen $u \equiv 0$ und $v \equiv 0$ für (D1) zulässig sind, sind sie eine Optimallösung von (D1). Nach dem Hauptsatz für duale lineare Programme /LUENBERGER-72/ existiert damit auch eine optimale also insbesondere zulässige Lösung von (P).

Es sei nun $t_1 : T \longrightarrow \mathbb{R}$, sodaß:

$$(+)\quad \forall (x,y) \in R: \quad \lambda(x,y) \leq t_1(y) - t_1(x) \leq \kappa(x,y).$$

Diese Abbildung ist nicht notwendig ganzzahlig und positiv.

Behauptung: $t_2 : T \longrightarrow \mathbb{Z}$ mit $t_2(x) := \lceil t_1(x) \rceil$ erfüllt ebenfalls (+). Hierbei bezeichnet $\lceil t(x) \rceil$ die kleinste ganze Zahl größer-gleich $t(x)$.

Es $(x,y) \in R$:

$$t_1(y) - t_1(x) \leq \kappa(x,y) \Rightarrow t_1(y) \leq t_1(x) + \kappa(x,y) \Rightarrow$$

$$\lceil t_1(y) \rceil \leq \lceil t_1(x) + \kappa(x,y) \rceil = \lceil t_1(x) \rceil + \kappa(x,y) \quad (* \text{ da } \kappa(x,y) \in \mathbb{N}^*) \Rightarrow$$

$$t_2(y) - t_2(x) = \lceil t_1(y) \rceil - \lceil t_1(x) \rceil \leq \kappa(x,y)$$

$$\lambda(x,y) \leq t_1(y) - t_1(x) \Rightarrow \lambda(x,y) + t_1(x) \leq t_1(y) \Rightarrow$$

$$\lambda(x,y) + \lceil t_1(x) \rceil = \lceil \lambda(x,y) + t_1(x) \rceil \leq \lceil t_1(y) \rceil \Rightarrow$$

$$\lambda(x,y) \leq \lceil t_1(y) \rceil - \lceil t_1(x) \rceil = t_2(y) - t_2(x)$$

Die Funktion $t_2 : T \longrightarrow \mathbb{Z}$ ist zwar ganzzahlig, jedoch nicht notwendig positiv.

Der gerichtete Graph (T, R) bestehe aus $n \in \mathbb{N}$ Zusammenhangskomponenten:

$$T = T^1 + \dots + T^n \quad \text{und} \quad R = R(T^1) + \dots + R(T^n)$$

Damit gibt es für jedes x aus T genau eine Komponente T_i mit $x \in T_i$.

Definiere nun:

$$\forall j \in \{1, \dots, n\}: \quad \varepsilon_j := \min \{t_2(x) : x \in T_j\}$$

Damit kann eine Abbildung $t : T \longrightarrow \mathbb{N}$ definiert werden durch:

$$\forall j \in \{1, \dots, n\}: \quad (x \in T_j \Rightarrow t(x) := t_2(x) - \varepsilon_j + 1)$$

Die Positivität von t :

Sei $x \in T$. Dann gibt es genau ein $j \in \{1, \dots, n\} : x \in T_j$. Somit gilt:

$$t(x) = t_2(x) - \varepsilon_j + 1 \geq \min \{t_2(x) : x \in T_j\} - \varepsilon_j + 1 = \varepsilon_j - \varepsilon_j + 1 = 1$$

Zeige nun, daß t die Nebenbedingungen (a) und (b) erfüllt:

Es $(x, y) \in R$:

Es gibt genau ein $j \in \{1, \dots, n\} : (x, y) \in R(T_j)$

$$t(x) + \lambda(x, y) = t_2(x) - \varepsilon_j + 1 + \lambda(x, y) \leq t_2(y) - \varepsilon_j + 1 = t(y)$$

$$t(x) + \kappa(x, y) = t_2(x) - \varepsilon_j + 1 + \kappa(x, y) \geq t_2(y) - \varepsilon_j + 1 = t(y)$$

Damit ist die Existenz einer Abbildung $t : T \longrightarrow \mathbb{N}$ gezeigt.

"nur dann".

Ann.: Es gibt einen Zyklus (K, f) mit $f(1) = x$ und $f(K) = y$ in $(T, \text{sym}(R))$ mit negativer Bewertung:

$$c(K, f) + c(y, x) < 0 \Rightarrow c(y, x) < -c(K, f).$$

$$R_{K, f} := \{(f(j), f(j+1)) : j = 1, \dots, |K|-1\} \quad (* \text{ Pfeile von } (K, f) *)$$

1. Fall: $(y, x) \in R$:

$$c(y, x) = \kappa(y, x)$$

Da $t : T \longrightarrow \mathbb{N}$ die Nebenbedingungen (a) und (b) erfüllt, gilt folgende Ableitung:

$$t(x) - t(y) \leq \kappa(y, x) = c(y, x) < -c(K, f) =$$

$$-\sum_{(u,v) \in R_{K,f}} c(u,v) = -\sum_{(u,v) \in R_{K,f} \cap R} c(u,v) - \sum_{(u,v) \in R_{K,f} \cap d(R)} c(u,v) = -\sum_{(u,v) \in R_{K,f} \cap R} \kappa(u,v) + \sum_{(u,v) \in R_{K,f} \cap d(R)} \lambda(v,u) \leq$$

$$\sum_{(u,v) \in R_{K,f} \cap R} t(u) - t(v) - \sum_{(u,v) \in R_{K,f} \cap d(R)} t(u) - t(v) = \sum_{(u,v) \in R_{K,f}} t(u) - t(v) = t(f(1)) - t(f(K)) = t(x) - t(y)$$

Dies ist jedoch ein Widerspruch.

2. Fall $(y, x) \in d(R)$:

Somit gilt: $(x, y) \in R$.

$$c(y, x) = -\lambda(x, y)$$

$$c(K, f) + c(y, x) < 0 \Rightarrow \lambda(x, y) > c(K, f).$$

Da $t: T \longrightarrow N$ die Nebenbedingungen (a) und (b) erfüllt, gilt folgende Ableitung:

$$t(y) - t(x) \geq \lambda(x, y) > c(K, f) =$$

$$\sum_{(u,v) \in R_{K,f}} c(u,v) = \sum_{(u,v) \in R_{K,f} \cap R} c(u,v) + \sum_{(u,v) \in R_{K,f} \cap d(R)} c(u,v) = \sum_{(u,v) \in R_{K,f} \cap R} \kappa(u,v) + \sum_{(u,v) \in R_{K,f} \cap d(R)} -\lambda(v,u) \geq$$

$$\sum_{(u,v) \in R_{K,f} \cap R} t(v) - t(u) - \sum_{(u,v) \in R_{K,f} \cap d(R)} t(v) - t(u) = \sum_{(u,v) \in R_{K,f}} t(v) - t(u) = t(f(K/)) - t(f(1)) = t(y) - t(x)$$

Dies ist jedoch ein Widerspruch.

Damit ist der erste und umfangreichste Beweispunkt abgeschlossen.

Zeige nun für $t: T \longrightarrow N$ die Nebenbedingungen (c) und (d):

Zu Nebenbedingung (c):

Zeige nun, daß T_α für $1 \leq \alpha \leq t_{\max}$ bezüglich $t(R)$ stabil ist.

Ann.: $\exists x, y \in T_\alpha: (x, y) \in t(R)$.

Damit gibt es eine Kette (K, f) von x nach y in R mit $x \neq y$.

Dann gilt: $\forall j = 1, \dots, K/-1: t(f(j)) \leq t(f(j)) + \lambda(f(j), f(j+1)) \leq t(f(j+1))$

Nach Definition von T_α folgt:

$$\alpha < t(x) + \tau(x) \leq t(x) + \lambda(x, f(2)) \leq t(f(2)) \leq \dots \leq t(f(K/)) = t(y) \leq \alpha$$

Dies ist ein Widerspruch. Damit ist T_α stabil in $t(R)$ und es gilt wegen (ii) die Nebenbedingung (c) von RCS.

Zu Nebenbedingung (d):

Es sei $x \in T$. Dann gibt es genau ein $j \in \{1, \dots, n\}: x \in T_j$.

Nach Definition von t gibt es ein $y \in T_j$, sodaß $t(y) = 1$. Da $(T_j, R(T_j))$ zusammenhängend ist, gibt es eine Kette (K, f) in $\text{sym}(R(T_j))$ von x nach y . Da t die Nebenbedingungen (a) und (b) erfüllt, gilt mit $R^* := \{(f(j), f(j+1)): j = 1, \dots, n\}$:

$$\sum_{(u,v) \in R^*} c(u,v) = \sum_{(u,v) \in R^* \cap R} c(u,v) + \sum_{(u,v) \in R^* \cap d(R)} c(u,v) = \sum_{(u,v) \in R^* \cap R} \kappa(u,v) + \sum_{(u,v) \in R^* \cap d(R)} \lambda(v,u) \geq$$

$$\sum_{(u,v) \in R^* \cap R} t(v) - t(u) - \sum_{(u,v) \in R^* \cap d(R)} t(v) - t(u) = \sum_{(u,v) \in R^*} t(v) - t(u) = t(f(K)) - t(f(1)) = t(y) - t(x)$$

Damit folgt mit $c_{\max} := \max \{ c(u,v) : (u,v) \in \text{sym}(R) \}$:

$$t(x) \leq \sum_{(u,v) \in R^*} c(u,v) + t(y) \leq \sum_{(u,v) \in R^*} c_{\max} + 1 = c_{\max} * (/K/ - 1) + 1 \leq$$

$$c_{\max} * (/T/ - 1) + 1 \leq c_{\max} * (/T/ - 1) + 1 \leq t_{\max}$$

Damit ist t eine zulässige Lösung von RCS. Wegen Nebenbedingung (d) gibt es nur endlich viele zulässige Lösungen, nämlich höchstens $t_{\max}^{/T/}$. Somit existiert auch eine optimale Lösung.

Dies beendet den Beweis.

Die Bedingung (i) ist eine Charakterisierung des Ungleichungssystems (a) und (b). Die Bedingung (ii) besagt, daß für jede Teilmenge S von T , die parallel ausführbar sind, die von S angeforderten Ressourcen im System insgesamt verfügbar sind. Jedoch kann eine Lösung von RCS auch erzwingen, daß Elemente von S sequentiell ausgeführt werden. Deshalb ist die Bedingung (ii) nicht notwendig für Nebenbedingung (c).

Es wird nun ein wichtiger Spezialfall betrachtet:

Falls die Abbildungen $\tau : X \longrightarrow \mathbb{N}$ und $\mu : T \longrightarrow \mathbb{N}_0^k$ konstant sind, spricht man von Planungen für *Einheitsoperationen* (*unit task scheduling*). Ohne Einschränkung kann dann τ als konstant 1 angenommen werden.

Falls es genau eine Resource gibt (d.h. $/\Phi/ = 1$) und die Abbildung $\mu : T \longrightarrow \mathbb{N}_0^1$ konstant 1 ist und die Abbildungen λ bzw κ identisch 1 bzw ∞ sind, liegen Planungen für *identische Prozessoren ohne weitere Ressourcen* vor. Der Wert s gibt dann die Anzahl der Prozessoren an.

6. Definition.

Das Planungsproblem für *Einheitsoperationen auf identischen Prozessoren* (UTS) "unit task scheduling" wird folgendermaßen definiert:

Gegeben seinen $T, R, \tau=1, k=1, \mu=1, s, t_{\max} \in \mathbb{N}$

Gesucht ist eine Abbildung $t: T \longrightarrow \mathbb{N}$

Zielfunktion: $\max \{t(x) + 1 : x \in T\} \longrightarrow \min$

Nebenbedingungen:

$$(a) \forall (x, y) \in R: \quad t(x) + 1 \leq t(y)$$

$$(b) \forall \alpha = 1, \dots, t_{\max}: \quad T_{\alpha} := \{x \in T : t(x) = \alpha\} = t^{-1}(\alpha)$$

$$\forall \alpha = 1, \dots, t_{\max}: \quad |T_{\alpha}| \leq s$$

$$(c) \forall x \in T: \quad t(x) \leq t_{\max} \quad \text{enddef}$$

Die κ -Nebenbedingung wurde weggelassen, da sie trivialerweise erfüllt ist.

Obwohl dieses Problem die einfachste Variante von RCS ist, ist die Lösungskomplexität allgemein NP-vollständig /ULLMAN-75/. Falls die azyklische Relation R auf T eine ganz bestimmte Form hat, ergeben sich weitere Spezialfälle. Eine Liste der Komplexitätsstufen für Planungsaufgaben findet man in /COFFMAN-76/ pp 20,21. Ein neueres Resultat findet man in /PAPADIM-79/. Die bei der Generierung von Parallelarchitekturen entstehenden Planungsprobleme sind von der Art RCS. Deshalb werden UTS-Aufgaben nur am Rande noch weiter bearbeitet. Die Hauptaufgabe ist die Untersuchung von RCS.

5. Lösungssätze von RCS

In diesem Kapitel werden Lösungsmöglichkeiten für RCS untersucht. Dabei steht die Zusammensetzung von Teillösungen zu einer Gesamtlösung von RCS im Vordergrund. Um Teillösungen komfortabel aneinanderreihen zu können, wird die Problemstellung RCS noch einmal zu GRCS verallgemeinert. Dabei werden für jede Elementaroperation frühest mögliche und spätest mögliche Anfangszeitpunkte festgelegt. Für GRCS wird dann ein Existenzsatz bewiesen. Falls eine Teilmenge der Quellen von R zum frühest möglichen Zeitpunkt ihre Ausführung beginnt, bleibt für die restlichen Elementaroperationen eine Problemstellung GRCS' zurück. Falls eine zulässige (optimale) Lösung von GRCS' gegeben ist, wird in Satz 5 (6) bewiesen, wann die zusammengesetzte Lösung zulässig (optimal) für GRCS ist. Weiters werden Möglichkeiten angegeben, ob und wie Vereinfachungen für UTS auf GRCS übertragen werden können. Falls die Restriktion der Gesamtresourcen konstant ist, gibt der Satz 12 eine Normalform an, auf die jede zulässige Lösung von GRCS gebracht werden kann.

Wie schon in Kapitel 4 angesprochen, ist RCS nur durch vollständige Enumeration aller möglichen Lösungen zu lösen. Dies wird algorithmisch mit einem "branch and bound" Verfahren erfolgen. Zuerst muß die RCS auf die Bedürfnisse des b&b-Verfahrens zugeschnitten werden. Gegeben seien die Bezeichnungen aus (4.3). Es ergeben sich jedoch zwei Veränderungen:

- (i) Es werden zwei Funktionen $rel, due : T \longrightarrow N$ eingeführt. Die Funktion rel (due) gibt für die Elemente von T den frühest (spätest) möglichen Ausführungsbeginn an. Somit wird nun nicht nur der relative Abstand von Elementaroperationen berücksichtigt, sondern es werden auch absolute Zeitschranken eingeführt.
- (ii) Die Gesamtverfügbarkeiten der Ressourcen sind nun zeitabhängig.

Damit kann die Planungsaufgabe mit Ressourcenbeschränkungen und "release/ due"-Zeiten definiert werden:

1. Definition. (GRCS "generalized RCS")

Gegeben seien die Bezeichnungen aus (4.3) zusammen mit $\text{rel}, \text{due} : T \longrightarrow \mathbb{N}$ und $s : \{1, \dots, t_{\max}\} \longrightarrow \mathbb{N}_0^k$ und es gelte:

- (#) Die Abbildung s ist in allen Komponenten monoton steigend; d.h.
 $\forall l=1, \dots, k \quad \forall 1 \leq i < j \leq t_{\max} : s(i)_l \leq s(j)_l$

Gesucht ist eine Abbildung $t : T \longrightarrow \mathbb{N}$ mit

$$\max \{t(x) + \tau(x) : x \in T\} \longrightarrow \min$$

unter folgenden Nebenbedingungen:

- (a) $\forall (x, y) \in R : \quad t(x) + \lambda(x, y) \leq t(y)$
 (b) $\forall (x, y) \in R : \quad t(x) + \kappa(x, y) \geq t(y)$
 (c) $\forall \alpha = 1, \dots, t_{\max} : \quad T_\alpha := \{x \in T : t(x) \leq \alpha < t(x) + \tau(x)\}$
 $\forall \alpha = 1, \dots, t_{\max} : \quad \sum_{x \in T_\alpha} \mu(x) \leq s(\alpha)$
 (d) $\forall x \in T : \quad t(x) \leq t_{\max}$
 (e) $\forall x \in T : \quad t(x) \geq \text{rel}(x)$
 (f) $\forall x \in T : \quad t(x) \leq \text{due}(x)$

Das Problem GRCS wird mit dem 10-Tupel $(T, R, \tau, \mu, \kappa, \lambda, s, \text{rel}, \text{due}, t_{\max})$ bezeichnet.

Schreibweise: Es sei $A \subset T$. Dann bezeichnet $\mu(A)$ die Summe aller Anforderungen der Elementaroperationen aus A ; d.h.:

$$\mu(A) := \sum_{z \in A} \mu(z)$$

Die Umformulierung von RCS zu GRCS wurde vorgenommen, um Teillösungen komfortabel aneinanderreihen zu können. Es folgt nun ein Existenzsatz analog zu (4.5). Hierzu muß die Menge der Elementaroperationen um eine triviale Operation x^* vermehrt werden, wobei gilt:

$$(i) \quad \mu(x^*) = 0 \in \mathbb{N}_0^k$$

$$(ii) \quad \tau(x^*) = 1$$

Auf $T^* := T \cup \{x^*\}$ wird die Ordnung $R^* := R \cup (\{x^*\} \times T)$ definiert. Die Operation x^* wird vor jeder Operation aus T ausgeführt.

2. Existenzsatz.

Gegeben sei GRCS und (T^*, R^*) . Es gelte:

(i) In $(T^*, \text{sym}(R^*))$ mit der Pfeilbewertung $c : \text{sym}(R^*) \longrightarrow \mathbb{Z}$ mit

$$c(x, y) := \begin{cases} \text{due}(y) & \text{für } x = x^* \\ \kappa(x, y) & \text{für } (x, y) \in R \\ -\text{rel}(x) & \text{für } y = x^* \\ -\lambda(y, x) & \text{für } (x, y) \in d(R) \end{cases}$$

gibt es keinen negativ bewerteten Zyklus.

(ii) $\forall S \subset T$ $t(R)$ -stabil: $\sum_{x \in S} \mu(x) \leq s$

(iii) $t_{\max} \geq \max \{\text{due}(x) : x \in T\}$

Dann gibt es eine Optimallösung von GRCS.

Beweis: Betrachte (T^*, R^*) mit den Pfeilbewertungen $\kappa^*, \lambda^* : R^* \longrightarrow \mathbb{N}$

$$\lambda^*(x, y) := \begin{cases} \text{rel}(y) & \text{falls } x = x^* \\ \lambda(x, y) & \text{sonst} \end{cases}$$

$$\kappa^*(x, y) := \begin{cases} \text{due}(y) & \text{falls } x = x^* \\ \kappa(x, y) & \text{sonst} \end{cases}$$

Benütze nun den ersten Beweispunkt von (4.5):

Es gibt genau dann ein $t^* : T^* \longrightarrow \mathbb{N}$, sodaß gilt:

$$\forall (x, y) \in R^* : \quad \lambda^*(x, y) \leq t^*(y) - t^*(x) \leq \kappa^*(x, y),$$

wenn die Bedingung (i) gilt:

"immer dann":

$(T^*, \text{sym}(R^*))$ mit Pfeilbewertung $c^* : \text{sym}(R^*) \longrightarrow \mathbb{Z}$

$$c(x, y) := \begin{cases} \kappa^*(x, y) & \text{für } (x, y) \in R^* \\ -\lambda^*(x, y) & \text{für } (x, y) \in d(R^*) \end{cases}$$

hat wegen

$$\text{sym}(R^*) = \text{sym}(R) + (\{x^*\} \times T) + (T \times \{x^*\})$$

keinen negativ bewerteten Zyklus.

Behauptung: $t : T \longrightarrow \mathbb{N}$ mit $t(x) := t^*(x) - t^*(x^*)$ ist eine zulässige Lösung von GRCS.

Begründung:

- Positivität: $\forall x \in T: \quad t(x) = t^*(x) - t^*(x^*) \geq \lambda^*(x^*, x) = \text{rel}(x) > 0$
- Nebenbedingung (a): Sei $(x, y) \in R$:

$$t(x) + \lambda(x, y) = t^*(x) + \lambda^*(x, y) - t^*(x^*) \leq t^*(y) - t^*(x^*) = t(y)$$
- Nebenbedingung (b): Sei $(x, y) \in R$:

$$t(x) + \kappa(x, y) = t^*(x) + \kappa^*(x, y) - t^*(x^*) \geq t^*(y) - t^*(x^*) = t(y)$$
- Nebenbedingung (c): T_0 ist $t(R)$ stabil (* wie im Beweis von (4.5) *)
- Nebenbedingung (d): $\forall x \in T: \quad t(x) = t^*(x) - t^*(x^*) \leq \kappa^*(x, y) = \text{due}(x) \leq t_{\max}$
- Nebenbedingung (e): $\forall x \in T: \quad t(x) = t^*(x) - t^*(x^*) \geq \lambda^*(x^*, x) = \text{rel}(x)$
- Nebenbedingung (f): $\forall x \in T: \quad t(x) = t^*(x) - t^*(x^*) \leq \kappa^*(x, y) = \text{due}(x)$

Damit ist t eine zulässige Lösung von GRCS. Da es nur endlich viele, nämlich höchstens $t_{\max}^{|T|}$, Abbildungen von $T \longrightarrow \mathbb{N}$ gibt, gibt es auch eine Optimallösung von GRCS.

"nur dann":

Falls es in $\text{sym}(R^*)$ einen negativ bewerteten Zyklus gibt, gibt es (nach Beweis von (4.5)) keine Abbildung $t^*: T^* \longrightarrow \mathbb{N}$ mit

$$(+)\quad \forall (x,y) \in R^*: \quad \lambda^*(x,y) \leq t^*(y) - t^*(x) \leq \kappa^*(x,y).$$

Ann.: Es gibt eine zulässige Lösung $t: T \longrightarrow \mathbb{N}$ von GRCS.

Definiere $t^*: T^* \longrightarrow \mathbb{N}$ mit

$$t^*(x) := \begin{cases} t(x) + 1 & \text{für } x \neq x^* \\ 1 & \text{für } x = x^* \end{cases}$$

Zeige nun, daß t^* die Bedingung (+) erfüllt.

Sei $(x,y) \in R^*$

1. Fall: $x = x^*$:

$$\lambda^*(x,y) = \text{rel}(y) \leq t(y) = t(y) + 1 - 1 = t^*(y) - t^*(x) = t(y) \leq \text{due}(y) = \kappa^*(x,y).$$

2. Fall: $x \neq x^*$:

$$\lambda^*(x,y) = \lambda(x,y \leq t(y) - t(x) = t(y) + 1 - (t(x) + 1) = t^*(y) - t^*(x) = t(y) - t(x) \leq \kappa(x,y) = \kappa^*(x,y).$$

Damit erfüllt t^* die Bedingung (+). - Widerspruch -

Somit gibt es, falls es in $\text{sym}(R^*)$ einen negativ bewerteten Zyklus gibt, keine zulässige Lösung von GRCS. qed.

Es wird eine Vereinfachung von GRCS vorbereitet. Sie dient dazu, die Knotenzahl des zu lösenden GRCS zu verringern, ohne auf die Optimalität der Lösung verzichten zu müssen.

Es sei nun $a^* := \min \{ \text{rel}(x) : x \in Q_R \}$ der frühest mögliche Anfangszeitpunkt für die Quellen von R (das sind die einzigen Elemente von R , die zur Ausführung anstehen). Die Elemente aus Q_R , für die das Minimum angenommen wird, werden mit Q^* bezeichnet. Somit ist

$$Q^* := \text{rel}^{-1}(a^*) \cap Q_R$$

Es wird nun eine nichtleere Teilmenge $A \subset Q^*$ betrachtet. Die Menge A wird im folgenden die Elementaroperationen darstellen, die zum Zeitpunkt α^* ihre Ausführung beginnen. Nun wird auf der Teilrelation $T' := T \setminus A, R' := R(T')$ ein GRCS' betrachtet. Die Abbildungen τ', μ', λ' und κ' ergeben sich als Restriktionen $\tau|_{T'}, \mu|_{T'}, \lambda|_{R'}$ und $\kappa|_{R'}$. Für rel', due' und s' ergibt sich folgende

3. Updatingvorschrift.

Definiere nun $\forall \alpha = 1, \dots, t_{\max}: A_\alpha := \{x \in A : \alpha^* \leq \alpha < \alpha^* + \tau(x)\}$.

Damit kann die Abbildung $s' : \{1, \dots, t_{\max}\} \longrightarrow N_0^k$ definiert werden durch:

$$s'(\alpha) := \begin{cases} s(\alpha) - \sum_{z \in A_\alpha} \mu(z) & \text{für } \alpha \geq \alpha^* \\ 0 & \text{sonst} \end{cases}$$

Die Abbildung $\text{rel}', \text{due}' : T' \longrightarrow N$ werden definiert durch

$$\text{rel}'(x) := \begin{cases} \max(\{\alpha^* + \lambda(z, x) : z \in V(x) \cap A\} \cup \{\text{rel}(x)\}) & \text{für } x \in N(A) \\ \alpha^* + 1 & \text{für } x \in Q^* \setminus A \\ \text{rel}(x) & \text{sonst} \end{cases}$$

$$\text{due}'(x) := \begin{cases} \min(\{\alpha^* + \kappa(z, x) : z \in V(x) \cap A\} \cup \{\text{due}(x)\}) & \text{für } x \in N(A) \\ \text{due}(x) & \text{sonst} \end{cases}$$

Das folgende Lemma zeigt, daß GRCS' wohldefiniert ist. Dazu muß die Monotonie von s' gezeigt werden.

4. Lemma.

Gegeben sei GRCS aus (1) und die Updatingvorschrift (3). Falls

$$\sum_{z \in A} \mu(z) \leq s(\alpha^*),$$

so ist s' nicht negativ und monoton steigend.

Beweis:

Die Mengen A_α bilden für $\alpha \geq \alpha^*$ eine monoton fallende Mengenfolge, denn es sei $\alpha_1 \geq \alpha_2 \geq \alpha^*$: $\{x \in A : \alpha^* + \tau(x) < \alpha_1\} \subset \{x \in A : \alpha^* + \tau(x) < \alpha_2\}$.

Damit ist die Funktion $\Delta s : \{\alpha^*, \dots, t_{\max}\} \longrightarrow N_0^k$

$$\Delta s(\alpha) := \sum_{z \in A_\alpha} \mu(z)$$

in jeder Komponente monoton fallend, da μ in jeder Komponente nicht negativ ist.

Damit hat die Funktion s' stets nicht negative Werte, denn (für $\alpha < \alpha^*$ ist es offensichtlich) sei $\alpha \geq \alpha^*$. Die Funktion $-\Delta s : \{\alpha^*, \dots, t_{\max}\} \longrightarrow N_0^k$ ist monoton steigend und somit auch die Funktion $s - \Delta s : \{\alpha^*, \dots, t_{\max}\} \longrightarrow N_0^k$. Da $s - \Delta s$ in α^* nach Voraussetzung nicht negativ ist, ist sie dies auch für alle Werte $\alpha \geq \alpha^*$.

Da $s - \Delta s : \{\alpha^*, \dots, t_{\max}\} \longrightarrow N_0^k$ monoton steigend und nicht negativ ist, ist dies auch die Fortsetzung durch Null auf der Menge $\{1, \dots, \alpha^* - 1\}$. qed

Im nächsten Lemma wird unter gewissen Voraussetzungen gezeigt, wie aus einer zulässigen Lösung von GRCS' eine zulässige Lösung von GRCS konstruiert werden kann. Hierbei ist jedoch entscheidend, daß eine zulässige Lösung von GRCS' existiert.

5. Lemma.

Gegeben sei $GRCS = (T, R, \tau, \mu, \kappa, \lambda, s, \text{rel}, \text{due}, t_{\max})$ und $GRCS' = (T', R', \tau', \mu', \kappa', \lambda', s', \text{rel}', \text{due}', t_{\max})$ zu nichtleerem $A \subset Q^*$. Es sei $t' : T' \longrightarrow N$ eine zulässige Lösung von $GRCS'$.

Dann ist $t : T \longrightarrow N$

$$t(x) := \begin{cases} \alpha^* & \text{für } x \in A \\ t'(x) & \text{für } x \in T' \end{cases}$$

genau dann eine zulässige Lösung von GRCS, wenn gilt:

$$(i) \quad \sum_{x \in A} \mu(x) \leq s(\alpha^*)$$

$$(ii) \quad \forall x \in A: \alpha^* \leq \text{due}(x)$$

$$(iii) \quad \alpha^* \leq t_{\max}$$

Beweis:

"immer dann": Es wird gezeigt, daß die Abbildung $t: T \longrightarrow N$ eine zulässige Lösung von GRCS ist.

zu Nebenbedingung (a) bzw (b):

Es sei $(x, y) \in R$.

Falls $(x, y) \in R'$ folgt $x, y \in T'$ und damit gilt:

$$t(x) + \lambda(x, y) = t'(x) + \lambda'(x, y) \leq t'(y) = t(y)$$

$$t(x) + \kappa(x, y) = t'(x) + \kappa'(x, y) \geq t'(y) = t(y)$$

Falls nun $(x, y) \in R \setminus R'$ folgt, daß $x \in A \subset Q_R$ und $y \in N_R(x) \subset T'$, da A als Teilmenge der Quellen von R stabil in R ist.

$$t(x) + \lambda(x, y) = \alpha^* + \lambda(x, y) \leq \text{rel}'(y) \leq t'(y) = t(y)$$

Die erste Ungleichung gilt mit Updatingvorschrift (3), die zweite wegen der Zulässigkeit von t' für GRCS'.

$$t(x) + \kappa(x, y) = \alpha^* + \kappa(x, y) \geq \text{due}'(y) \geq t'(y) = t(y)$$

Die erste Ungleichung gilt wegen (3), die zweite wegen der Zulässigkeit von t' für GRCS'.

zu Nebenbedingung (c):

$$T_\alpha := \{x \in T: t(x) \leq \alpha < t(x) + v(x)\}$$

$$T'_\alpha := \{x \in T': t'(x) \leq \alpha < t'(x) + v(x)\}$$

$$A_\alpha := \{x \in A: \alpha^* \leq \alpha < \alpha^* + v(x)\}$$

Es sei $\alpha \in \{1, \dots, t_{\max}\}$. Dann gilt: $T_\alpha = T'_\alpha \cup A_\alpha$, denn $T_\alpha = (T_\alpha \cap T') \cup (T_\alpha \cap A)$
 $= T'_\alpha \cup A_\alpha$

Für alle $\alpha < \alpha^*$ sind alle $T_\alpha = \emptyset$. Es sei nun $\alpha \geq \alpha^*$

$$\sum_{x \in T_a} \mu(x) = \sum_{x \in T'_a} \mu(x) + \sum_{x \in A_a} \mu(x) = \sum_{x \in T'_a} \mu'(x) + \sum_{x \in A_a} \mu(x) \leq$$

$$s'(a) + \sum_{x \in A_a} \mu(x) = s(a) - \sum_{x \in A_a} \mu(x) + \sum_{x \in A_a} \mu(x) = s(a)$$

zu Nebenbedingung (d):

Sei $x \in T$. Falls $x \in T'$, gilt $t(x) \leq t_{\max}$, da $t|_{T'}$ eine zulässige Lösung von GRCS' darstellt. Sei nun $x \in A$. Dann gilt: mit (i) und (ii)

$$t(x) = \alpha^* \leq t_{\max} \quad (* \text{ nach Voraussetzung (iii) } *)$$

zu Nebenbedingung (e):

Sei $x \in T$. Falls $x \in A$ gilt: $t(x) = \alpha^* = \text{rel}(x)$

Falls $x \in T \setminus A$, folgt: $x \in T'$ und $\text{rel}'(x) \geq \text{rel}(x)$.

Mit $t|_{T'}$ zulässig für GRCS' gilt: $t(x) = t'(x) \geq \text{rel}'(x) \geq \text{rel}(x)$

zu Nebenbedingung (f):

Sei $x \in T$. Falls $x \in A$, gilt nach (ii): $t(x) = \alpha^* \leq \text{due}(x)$

Falls $x \in T \setminus A$, folgt: $x \in T'$ und $\text{due}'(x) \leq \text{due}(x)$.

Mit $t|_{T'}$ zulässig für GRCS' gilt: $t(x) = t'(x) \leq \text{due}'(x) \leq \text{due}(x)$.

Damit ist t eine zulässige Lösung für GRCS.

"nur dann": Falls die Bedingung (i) verletzt ist, ist für t die Nebenbedingung (c) zum Zeitpunkt α^* verletzt. Falls die Bedingung (ii) nicht gilt, erfüllt die Abbildung t nicht die Nebenbedingung (f). Falls die Bedingung (iii) nicht gilt, erfüllt t nicht die Nebenbedingung (d). qed.

Man sieht sofort, daß eine zulässige Lösung von GRCS' nur dann existiert, wenn $\{x \in Q^* : \text{due}(x) = \alpha^*\} \subset A$ ist.

Falls die Lösung t' von GRCS' optimal war, gibt der folgende Satz Bedingungen an, bei deren Erfüllung die in Lemma 5 konstruierte Lösung t eine Optimallösung von GRCS ist.

6. Satz

Gegeben sei GRCS und GRCS' zu $A = Q^*$ und $\alpha_{\max} := \max \{\alpha^* + \tau(x) : x \in Q^*\}$. Es sei $\alpha^* \leq t_{\max}$ und $\forall x \in A : \alpha^* \leq \text{due}(x)$. Es sei weiters $t' : T' \longrightarrow N$ eine Optimallösung von GRCS'.

Die Abbildung $t : T \longrightarrow N$

$$t(x) := \begin{cases} \alpha^* & \text{für } x \in Q^* \\ t'(x) & \text{für } x \in T' \end{cases}$$

ist immer dann eine Optimallösung von GRCS, wenn gilt:

- (i) $\sum_{x \in Q^*} \mu(z) \leq s(\alpha^*)$
- (ii) $\forall \alpha = \alpha^* + 1, \dots, \alpha_{\max} - 1 : Q'_\alpha := \{x \in Q_{R'} : \text{rel}'(x) \leq \alpha < \text{rel}'(x) + \tau(x)\}$
 $\forall \alpha = \alpha^* + 1, \dots, \alpha_{\max} - 1 : \sum_{x \in Q'_\alpha} \mu(z) \leq s'(\alpha)$
- (iii) $\max \{\alpha^* + \tau(x) : x \in Q^*\} \leq \min \{\text{rel}'(x) + \tau(x) : x \in Q_{R'}\}$

Beweis:

Da t nach Lemma (5) eine zulässige Lösung von GRCS ist, existiert wegen der Endlichkeit des Lösungsraumes auch eine optimale Lösung t^* .

Zeige nun: $\max \{t(x) + \tau(x) : x \in T\} \leq \max \{t^*(x) + \tau(x) : x \in T\}$

Definiere nun $g : T' \longrightarrow N$ mit $g \equiv t^*|_{T'}$

Zeige : g ist eine zulässige Lösung von GRCS'.

Die Nebenbedingungen (a),(b) und (d) sind offensichtlich.

zu Nebenbedingung (c):

Sei $\alpha \in \{1, \dots, t_{\max}\}$. Definiere $T^*_\alpha := \{x \in T : t^*(x) \leq \alpha < t^*(x) + \tau(x)\}$ und

$T'^*_\alpha := \{x \in T \setminus Q^* : t^*(x) \leq \alpha < t^*(x) + \tau(x)\} = T^*_\alpha \setminus Q^*$

Falls $\alpha < \alpha^* : T^*_\alpha = \emptyset$ woraus Nebenbedingung (b) folgt.

Falls $\alpha = \alpha^* : T^*_\alpha \subset Q^*$, da $Q^* = \{x \in T : \text{rel}(x) \leq \alpha^*\}$. Somit ist T'^*_α leer. Damit gilt die Nebenbedingung (b).

Falls $\alpha \geq \max \{\alpha^* + \tau(x) : x \in Q^*\}$ folgt mit der Updatingvorschrift 3, daß

$Q^*_\alpha = T^*_\alpha \cap Q^*$ leer ist. Diese Vorschrift liefert auch für $A = Q^* : s(\alpha) = s'(\alpha)$.

Somit gilt:

$$\mu(T'^*_\alpha) = \mu(T'^*_\alpha \cap Q^*) + \mu(T'^*_\alpha \setminus Q^*) \leq \mu(T^*_\alpha) \leq s(\alpha) = s'(\alpha).$$

Falls nun $(\#) \alpha^* < \alpha < \max \{\alpha^* + \tau(x) : x \in Q^*\}$:

Behauptung: $T'^*_\alpha \subset Q_{R'}$

Begründung: Falls es ein $x \in T'^*_\alpha \setminus Q_{R'}$ gibt, gibt es ein $y \in Q_{R'}$ mit $(y, x) \in R'$.

Es gilt: $\alpha \geq t^*(x) \geq t^*(y) + \lambda(y, x) \geq \text{rel}'(y) + \lambda(y, x) \geq \text{rel}'(y) + \tau(y) \geq$

$\min \{ \text{rel}'(z) + \tau(z) : z \in Q_{R'} \} \geq \max \{ \alpha^* + \tau(x) : x \in Q^* \} > \alpha$

Die erste Ungleichung gilt, da $x \in T'^*_\alpha$, die zweite, da $(y, x) \in R'$ und die dritte wegen Nebenbedingung (e), welche anschließend gezeigt wird. Die vierte Ungleichung gilt, da $\forall (u, v) \in R : \tau(u) \leq \lambda(u, v)$ gilt, die fünfte, da $y \in Q_{R'}$, und die sechste wegen Voraussetzung (iii). Damit ergibt sich ein Widerspruch und es gilt:

$T'^*_\alpha \subset Q_{R'}$.

Es gilt sogar:

$T'^*_\alpha \subset Q'_\alpha$ (* definiert in (ii) *)

Begründung:

Es sei $x \in T'^*_\alpha \Rightarrow t^*(x) \geq \text{rel}'(x)$ da $x \in Q_{R'}$ und Nebenbedingung (e).

bleibt, zu zeigen: $t^*(x) < \text{rel}'(x) + \tau(x)$:

$t^*(x) \leq \alpha < \max \{ \alpha^* + \tau(x) : x \in Q^* \} \leq \min \{ \text{rel}'(z) + \tau(z) : z \in Q_{R'} \} \leq \text{rel}'(x) + \tau(x)$

Die erste Ungleichung gilt, da $x \in T'^*_\alpha$, die zweite wegen $(\#)$, und die dritte nach Voraussetzung (iii).

Somit gilt: $T'^*_\alpha \subset Q'_\alpha$. Hieraus folgt unmittelbar die Nebenbedingung (c) nach (ii):

$$\mu(T'^*_\alpha) \leq \mu(Q'_\alpha) \leq s'(\alpha)$$

zu Nebenbedingung (e): Sei $x \in T'$

Falls $x \in T' \setminus N(Q^*)$ gilt mit der Updatingvorschrift 3:

$$t^*(x) \geq \text{rel}(x) = \text{rel}'(x).$$

Falls $x \in N(Q^*)$ hat x mindestens einen Vorgänger in Q^* . Damit folgt:

$$t^*(x) \geq \max (\{t^*(y) + \lambda(y, x) : y \in V(x) \cap Q^*\} \cup \{\text{rel}(x)\}) =$$

$$\max (\{\alpha^* + \lambda(y, x) : y \in V(x) \cap Q^*\} \cup \{\text{rel}(x)\}) = \text{rel}'(x)$$

zu Nebenbedingung (f): Sei $x \in T'$

Falls $x \in T' \setminus N(Q^*)$ gilt mit der Updatingvorschrift 3:

$$t^*(x) \leq \text{due}(x) = \text{due}'(x)$$

Falls $x \in N(Q^*)$ hat x mindestens einen Vorgänger in Q^* . Damit folgt:

$$t^*(x) \leq \min (\{t^*(y) + \kappa(y, x) : y \in V(x) \cap Q^*\} \cup \{\text{due}(x)\}) =$$

$$\min (\{\alpha^* + \kappa(y, x) : y \in V(x) \cap Q^*\} \cup \{\text{due}(x)\}) = \text{rel}'(x)$$

Damit ist $g \equiv t^*|_{T'}$ eine zulässige Lösung von GRCS'.

1. Fall: $\max \{t(x) + \tau(x) : x \in T\} = \max \{t(x) + \tau(x) : x \in Q^*\}$ Dann gilt:

$$\max \{t(x) + \tau(x) : x \in T\} = \max \{t(x) + \tau(x) : x \in Q^*\} = \max \{\text{rel}(x) + \tau(x) : x \in Q^*\} \leq$$

(* $\text{rel}(x)$ ist der frühest mögliche Anfangszeitpunkt vergl. (e) *)

$$\max \{t^*(x) + \tau(x) : x \in Q^*\} \leq \max \{t^*(x) + \tau(x) : x \in T\}$$

2. Fall: $\max \{t(x) + \tau(x) : x \in T\} = \max \{t(x) + \tau(x) : x \in T \setminus Q^*\}$. Dann gilt:

$$\max \{t(x) + \tau(x) : x \in T\} = \max \{t(x) + \tau(x) : x \in T \setminus Q^*\} = \max \{g(x) + \tau(x) : x \in T\} \leq$$

(* g ist zulässig für GRCS' *)

$$\max \{t'(x) + \tau(x) : x \in T\} \leq \max \{t^*(x) + \tau(x) : x \in T\}$$

Dies beendet den Beweis von Satz 7.

Der Satz 7 gibt eine erste Möglichkeit, Elementaroperationen an der Spitze der Ordnung R zu entfernen, ohne die Optimalität der Lösung zu beeinflussen. Die Bedingung (i) ist offensichtlich, da alle Operationen aus Q^* zum Zeitpunkt α ihre Bearbeitung beginnen sollen. Dementsprechend müssen die geforderten Resour-

cen vorhanden sein. Die Bedingung (ii) besagt, daß prinzipiell (von der Bereitstellung der Ressourcen) alle Operationen aus Q_R zum frühest möglichen Zeitpunkt ihre Ausführung beginnen. Die Nebenbedingung (iii) besagt, daß sich die Operationen aus Q^* höchstens mit den Operationen aus Q_R zeitlich überlappen. Das folgende Gegenbeispiel zeigt, daß man auf die Bedingungen (ii) und (iii) nicht verzichten kann.

7. Beispiel.

Es seien 5 Elementaroperationen a, b, c, d, e und zwei Ressourcen ϕ_1 und ϕ_2 gegeben. Die Verfügbarkeiten der Ressourcen sind konstant mit $s = (1, 2)$. Der Wert $t_{\max}$ wird auf 5 festgelegt. Ein den Elementaroperationen zugeordneter Vektor v hat folgende Komponenten:

$$v(x) = (\text{rel}(x), \text{due}(x), \tau(x), \mu(x)_1, \mu(x)_2) \quad \text{für } x \in \{a, b, c, d, e\}$$

$$(1, 3, 2, 1, 1)$$

a 

$$(2, 3, 1, 1, 1)$$

b 

$$(1, 5, 1, 0, 1)$$

d 

$$(1, 5, 1, 0, 1)$$

e 

$$(1, 2, 1, 0, 1)$$

c 

$$Q_R = \{a, b, c\}, \alpha^* = 1, Q^* = \{a, c\}.$$

Die Bedingung (i) von Satz 7 ist erfüllt, denn:

$$\mu(Q^*) = (1, 1) + (0, 1) = (1, 2) \leq s$$

Falls die Elemente von Q^* ihre Ausführung zum Zeitpunkt $\alpha^* = 1$ beginnen, ergibt sich eine unter dieser Restriktion optimale Lösung:

Zeit	1	2	3	4	5
Operationen	a	a	b	d	e
		c			

Eine optimale Lösung kann jedoch folgende Gestalt haben:

Zeit	1	2	3	4	5
Operationen	c	b	d	e	
			a	a	

Die Eigenart dieses Beispiels ist es, daß Operation a die Resource ϕ_1 belegt, die im nächsten Zeittakt von Operation b benötigt wird. Durch die Ausführung von a werden nun b und alle seine Nachfolger blockiert, obwohl die Nachfolger die Resource ϕ_1 nicht benötigen.

8. Folgerung.

Es gelten die Voraussetzungen von Satz 6 und die Bedingung (i). Die in Satz 6 konstruierte Abbildung t ist eine optimale Lösung von GRCS, falls gilt:

$$(iv) \quad \max \{a^* + \tau(x) : x \in Q^*\} \leq \min \{\text{rel}'(x) : x \in Q_R\}$$

Beweis:

Der Beweis geht völlig analog zum Beweis von Satz 6. Die einzige Änderung ergibt sich beim Nachweis der Nebenbedingung (c) für die Restriktion von t^* auf $T \setminus Q^*$.

zu (c):

Falls $\alpha < \min \{\text{rel}'(x) : x \in Q_R\}$ folgt: $T^*_\alpha = \emptyset$ und somit (c).

Falls $\alpha \geq \min \{\text{rel}'(x) : x \in Q_R\}$ folgt mit (iv):

$$\alpha \geq \max \{a^* + \tau(x) : x \in Q^*\} \Rightarrow \forall x \in Q^* : \alpha \geq a^* + \tau(x) = t^*(x) + \tau(x).$$

Somit folgt, daß $Q^*_\alpha = \emptyset$. Hieraus folgt mit der Updatingvorschrift 3:

$$s(a) = s'(a) \text{ woraus sofort die Nebenbedingung (c) folgt.} \quad \text{qed}$$

Zwei Vereinfachungen für UTS (4.6) sind in /DOLEV-84/ und /DOLEV-85/ angegeben. Sie basieren auf folgender Tatsache:

Falls die Ordnung R leer ist und somit die Elementaroperationen unabhängig sind, ist die Lösung von UTS trivial. Zu jedem Zeitpunkt t werden s (falls überhaupt vorhanden) Operationen geplant. Die Gesamtausführungszeit ist dann $\lceil n/s \rceil$ mit $n := |T|$. Falls nun bei der Lösung von UTS unabhängige Elementaroperationen entstehen, werden sie aus der Planung entfernt und mit entsprechenden "release" Zeiten separat abgespeichert. Falls nun die "Restplanung" beendet ist, ergibt sich für jeden Zeitpunkt a ein Ressourcenverschnitt (untätige Prozessoren) der Größe $\Delta s(a) := s - |T_a|$. Die Planung der vorher entfernten unabhängigen Elementaroperationen auf dem Profil $\Delta s : \{1, \dots, t_{\max}\} \rightarrow \mathbb{N}_0$ ist trivial. Damit kann für eine vorgegebene Ordnung R (beschränkter maximaler Höhe) ein polynomial

beschränktes Verfahren für UTS definiert werden /DOLEV-84/.

Eine ähnliche Vereinfachung ergibt sich, wenn die Ordnung (auch erst im Verlauf des b&b Verfahrens) aus mehr als s Zusammenhangskomponenten besteht. Dort werden die Elementaroperationen vorerst nicht betrachtet, die zu Komponenten gehören, deren maximale Höhe kleiner als der "median" ist. Dieser Wert ist die maximale Höhe des s -höchsten Komponente in R . In /DOLEV-85/ wird gezeigt, daß die vorerst weggelassenen Elementaroperationen auf triviale Weise zur Planung der restlichen Operationen hinzugefügt werden können.

Nun soll untersucht werden, in wie weit die oben skizzierten Vereinfachungen auch für RCS anwendbar sind. Die grundlegende Idee von Dolev war, daß (nun in der Formulierung von GRCS) folgendes Problem trivial zu lösen war:

- $R = \emptyset$ unabhängige Elementaroperationen
- $\tau = 1$ Einheitsoperationen
- $\lambda = 1$ triviale untere Schranke des Zeitabstandes
- $\kappa = \infty$ triviale obere Schranke des Zeitabstandes
- $k = 1$ genau eine Resource (nämlich Prozessoren)
- $\mu = 1$ Jede Operation benötigt genau einen Prozessor.

Die Erwartung, eine Vereinfachung für GRCS erzielen zu können, erhalten mit der Arbeit von /GAREY-74/ einen entscheidenden Dämpfer: Dort wird nämlich gezeigt, daß folgendes Problem (in der Formulierung von GRCS) in der Klasse NP-vollständig ist.

- $R = \emptyset$ unabhängige Elementaroperationen
- $\tau = 1$ Einheitsoperationen
- $\lambda = 1$ triviale untere Schranke des Zeitabstandes
- $\kappa = \infty$ triviale obere Schranke des Zeitabstandes
- $k = 2$ zwei Ressourcen
- $\mu_1 = 1$ Prozessorresource
- μ_2 beliebig eine weitere Resource
- $s_1 = 3$ 3 Prozessoren
- s_2 beliebig Ressourcenverfügbarkeit

Auch die Idee eines "greedy" Verfahrens schlägt fehl. Ein greedy Verfahren löst in jedem Zeitschritt ein Verschnittproblem, wobei die Summe der nicht benötigten Ressourcen minimiert wird.

Das folgende Gegenbeispiel zeigt jedoch, daß für Einheitsoperationen eine Optimallösung nicht "greedy" zu sein braucht.

9. Beispiel.

Gegeben seien 8 unabhängige Einheitsoperationen $T := \{1, \dots, 8\}$ und $R = \emptyset$. Es stehen zwei Ressourcen zur Verfügung mit Restriktion $s := (4, 4)$. Die Elementaroperationen aus T haben folgende Anforderungen:

$\mu(i) := (1, 1)$ für $i = 1, \dots, 4$

$\mu(i) := (3, 1)$ für $i = 5, \dots, 8$

Eine Greedylösung sieht folgendermaßen aus:

Zeit	1	2	3	4	5
Operationen	1	5	6	7	8
	2				
	3				
	4				

Im ersten Zeitschritt ist der Verschnitt Null, da beide Ressourcen voll ausgeschöpft sind.

Für eine Optimallösung wird in keinem Zeitschritt ein Nullverschnitt erreicht und trotzdem ist diese Lösung besser als jede Greedylösung. Sie kann folgendes Aussehen haben:

Zeit	1	2	3	4	5
Operationen	1	2	3	4	
	5	6	7	8	

Wir betrachten kurz die Problemstellung UTS.

10. Lemma.

Gegeben sei UTS aus (4.6) und es sei $t^* : T \longrightarrow \mathbb{N}$ eine Optimallösung von UTS. Dann gibt es eine Optimallösung $t' : T \longrightarrow \mathbb{N}$ von UTS, für die gilt:

$$(+) \quad |t'^{-1}(1)| = \min(|Q_R|, s)$$

Beweis:

Vor dem eigentlichen Beweis von Lemma 11 soll es noch interpretiert werden. Die-

ses Lemma sagt aus, daß es für UTS stets eine Optimallösung gibt, die "greedy" ist. Im ersten Zeittakt der Planung wird der Verschnitt (Anzahl der nicht arbeitenden Prozessoren) minimiert. Damit müssen von den Elementaroperationen, die "ready for execution" sind, jeweils nur maximale Menge (bezüglich der Restriktion) im b&b Verfahren betrachtet werden.

Es sei t^* eine Optimallösung von UTS und $m := \min (/Q_R/, s)$.

Es gilt: $/t^{*-1}(1)/ \leq \min (/Q_R/, s)$, denn im ersten Zeittakt sind nur die Elemente aus Q_R "ready for execution". Damit gilt: $/t^{*-1}(1)/ \leq /Q_R/$. Andererseits ist t^* zulässig für UTS und erfüllt damit insbesondere die Nebenbedingung (b) für $\alpha = 1$ in (4.6): $/t^{*-1}(1)/ \leq s$. Somit gilt insgesamt: $/t^{*-1}(1)/ \leq \min (/Q_R/, s)$

Falls $/t^{*-1}(1)/ = \min (/Q_R/, s)$, muß nichts gezeigt werden.

Falls nun $/t^{*-1}(1)/ < \min (/Q_R/, s)$ gibt es ein $Q' \subset Q_R$ mit folgenden Eigenschaften:

$$(i) \quad /Q'/ = m - /t^{*-1}(1)/ =: n$$

$$(ii) \quad \forall x \in Q': \quad t^*(x) > 1$$

Q' kann als beliebige n elementige Menge von $Q_R \cap t^{*-1}(1+N)$ gewählt werden, denn $/Q_R \cap t^{*-1}(1+N)/ = /Q_R/ - /Q_R \cap t^{*-1}(1)/ \geq m - /Q_R \cap t^{*-1}(1)/ = m - /t^{*-1}(1)/ = n$

Definiere nun $t': T \longrightarrow N$

$$t'(x) := \begin{cases} 1 & \text{für } x \in Q' \\ t^*(x) & \text{sonst} \end{cases}$$

Zeige nun, daß t' zulässig für UTS ist:

Nebenbedingung (a): Sei $(x,y) \in R$, falls $(x,y) \in R(T \setminus Q')$ gilt (a), da t^* zulässig war.

Falls $(x,y) \in R \setminus R(T \setminus Q')$, folgt: $x \in Q', y \in T \setminus Q'$ und damit gilt:

$$t'(x) + 1 = 1 + 1 \leq t^*(x) + 1 \leq t^*(y) = t'(y)$$

Nebenbedingung (b): Falls $\alpha > 1$, gilt: $/t'^{-1}(\alpha)/ \leq /t^{*-1}(\alpha)/ \leq s$

$$\text{Falls } \alpha = 1, \text{ gilt: } /t'^{-1}(1)/ = /t^{*-1}(1)/ + /Q'/ = /t^{*-1}(1)/ + m - /t^{*-1}(1)/ = m \leq s$$

Nebenbedingung (c): Es gilt nach Konstruktion von t' :

$$\forall x \in T: t'(x) \leq t^*(x) \leq t_{\max}.$$

Damit ist t' eine zulässige Lösung. Da t' nie größere Werte als t^* annimmt, ist t' auch eine Optimallösung. qed

Die Übertragung von Lemma 10 auf GRCS kann wie das Beispiel 8 zeigt nicht ohne weiteres erfolgen. Die Schwierigkeit bei GRCS liegt nicht so sehr in der Vielfalt der Ressourcen sondern in der variablen Dauer ihrer Belegung.

Der wesentliche Gesichtspunkt des Beweises von Lemma 10 war der folgende: Falls die Menge $t^{-1}(1)$ nicht maximal war, konnten die "restlichen" Elementaroperationen zum Zeitpunkt 1 vorgezogen werden, denn sie hatten alle die Ausführungszeit 1. Somit ergibt sich eine unmittelbare Folgerung von Lemma 11:

11. Folgerung.

Gegeben sei GRCS mit α^* und Q^* zusammen mit einer Optimallösung t^* . Weiters sei $\kappa \equiv \infty$ und $Q^*_1 := \{x \in Q^*: t(x) = 1\}$.

Dann gibt es eine Optimallösung t von GRCS, für die gilt:

$$\forall x \in Q^*_1 \setminus t^{-1}(\alpha^*): \exists \phi_l \in \Phi: \mu(x)_l + \sum_{x \in t^{-1}(\alpha^*)} \mu(x) > s_l$$

Beweis:

Da die obere Schranke für den Zeitabstand unendlich ist, kann der Ausführungsbeginn einer Elementaroperation mit Ausführungsdauer 1 beliebig vorverlegt werden. Die Behauptung der Folgerung lautet informell:

Der Durchschnitt $t^{-1}(\alpha^*) \cap Q^*_1$ ist maximal in Q^*_1 bez. der Ressourcenbeschränkung (c) von GRCS. Der Beweis erfolgt völlig analog zum Beweis von Lemma 10.

Im folgenden Satz wird eine Normalform für Lösungen von GRCS bewiesen. Jede zulässige Lösung von GRCS kann ohne Erhöhung der Zielfunktion auf eine Form gebracht werden die folgende Bedingung erfüllt. Falls eine Elementaroperation x

erst nach ihrer Releasezeit ihre Ausführung beginnt, ist eine der drei folgenden Bedingungen erfüllt:

- Es gibt eine Elementaroperation y , deren Bearbeitungsende mit dem Bearbeitungsbeginn von x übereinstimmt.
- Es gibt einen Vorgänger y von x , sodaß der zeitliche Abstand des Ausführungsbeginns von x und y minimal (d.h. gleich $\lambda(y,x)$) ist.
- Es gibt einen Nachfolger y von x , sodaß der zeitliche Abstand des Bearbeitungsbeginns von x und y maximal (d.h. gleich $\kappa(x,y)$) ist.

Diesen Sachverhalt präzisiert der folgende

12. Satz.

Gegeben sei GRCS mit s konstant. Es sei $t : T \longrightarrow \mathbb{N}$ eine zulässige Lösung von GRCS. Dann gibt es eine weitere zulässige Lösung $t' : T \longrightarrow \mathbb{N}$ mit

$$(\#) \max \{ t'(x) + \tau(x) : x \in T \} \leq \max \{ t(x) + \tau(x) : x \in T \},$$

sodaß für alle $x \in T$ mit $t'(x) > \text{rel}(x)$ eine der folgenden 3 Alternativen gilt:

- (i) $\exists y \in T : \quad t(y) + \tau(y) = t(x)$
- (ii) $\exists y \in V(x) : \quad t(y) + \lambda(y,x) = t(x)$
- (iii) $\exists y \in N(x) : \quad t(y) - \kappa(x,y) = t(x)$

Beweis:

Der Beweis zeigt, wie die Abbildung t unter Erhalt der Zulässigkeit zu verändern ist, sodaß alternativ eine der Bedingungen (i),(ii),(iii) gilt.

Wähle nun x mit $t(x) = \min \{ t(z) : t(z) > \text{rel}(z) \text{ und } \neg((i) \vee (ii) \vee (iii)) \}$

Falls kein solches x existiert, ist nichts zu zeigen.

Somit gilt: $t(x) > \text{rel}(x)$ und

$$\forall y \in T \setminus A(x) : \quad t(y) + \tau(y) \neq t(x)$$

$$\forall y \in V(x): t(y) + \lambda(y, x) < t(x)$$

$$\forall y \in N(x): t(y) - \kappa(x, y) < t(x)$$

$$T_{<} := \{z \in T: t(z) + \tau(z) < t(x)\}$$

Setze nun

$$\alpha^+ := \max (\{t(z) + \tau(z) : z \in T_{<}\} \cup \{t(y) + \lambda(y, x) : y \in V(x)\} \cup \{t(y) - \kappa(x, y) : y \in N(x)\})$$

Falls $T_{<} \cup V(x) \cup N(x)$ leer ist, setze $\alpha^+ := -\infty$

Definiere nun eine Abbildung $t' : T \rightarrow \mathbb{N}$:

$$t'(z) := \begin{cases} \max \{\alpha^+, \text{rel}(x)\} & \text{für } z = x \\ t(z) & \text{sonst} \end{cases}$$

Zeige nun, daß t' eine zulässige Lösung von GRCS darstellt. Dazu müssen die Nebenbedingungen (a) - (f) aus (1) nachgewiesen werden:

(+) Zeige zunächst: $t'(x) < t(x)$ und t' erfüllt für x eine der Alternativen (i), (ii), (iii) oder $\text{rel}(x) = t'(x)$:

1. Fall: $t'(x) = \text{rel}(x)$. Dann gilt: $t'(x) = \text{rel}(x) < t(x)$.

2. Fall: $t'(x) = \alpha^+$. Dann gilt: $T_{<} \cup V(x) \cup N(x) \neq \emptyset$

Falls das Maximum für ein $y \in T_{<}$ angenommen wird, gilt:

$$t'(x) = \alpha^+ = \max \{t(z) + \tau(z) : z \in T_{<}\} < t(x), \text{ da } T_{<} \text{ endlich. Somit folgt (i).}$$

Falls das Maximum für ein $y \in V(x)$ angenommen wird, gilt:

$$t'(x) = \alpha^+ = \max \{t(y) + \lambda(y, x) : y \in V(x)\} < t(x), \text{ da } T_{<} \text{ endlich. Somit folgt (ii).}$$

Falls das Maximum für ein $y \in N(x)$ angenommen wird, gilt:

$$t'(x) = \alpha^+ = \max \{t(y) - \kappa(x, y) : y \in N(x)\} < t(x), \text{ da } T_{<} \text{ endlich. Somit folgt (iii).}$$

In jedem Fall gilt: $t'(x) < t(x)$

zu (a): Es sei $(u, v) \in R$. Falls $u \neq x \neq v$, ist t' mit t identisch und somit gilt (a).

$$\begin{aligned} \text{Falls } u = x: t'(u) + \lambda(u, v) &= t'(x) + \lambda(x, v) < t(x) + \lambda(x, v) = t(u) + \lambda(u, v) \leq t(v) \\ &= t'(v) \end{aligned}$$

Falls $v = x$, gilt $u \in V(x)$. Damit gilt:

$$t'(u) + \lambda(u, v) = t(u) + \lambda(u, v) \leq \max \{t(z) + \lambda(z, v) : z \in V(x)\} \leq t'(x) = t'(v)$$

zu (b): Es sei $(u, v) \in R$. Falls $u \neq x \neq v$, ist t' mit t identisch und somit gilt (b).

Falls $u = x$, gilt $v \in N(x)$. Damit gilt:

$$t'(v) - \kappa(u, v) = t(v) - \kappa(u, v) \leq \max \{ t(z) - \kappa(x, z) : z \in N(x) \} \leq t'(x) = t'(u)$$

Falls $v = x$, gilt:

$$t'(v) - \kappa(u, v) = t'(x) - \kappa(u, v) < t(x) - \kappa(u, v) = t(x) - \kappa(u, x) \leq t(u)$$

zu (c): Es sei $\alpha \in \{1, \dots, t_{\max}\}$. $T'_\alpha := \{z \in T : t'(z) \leq \alpha < t'(z) + \tau(z)\}$

Falls $\alpha \geq t(x)$:

Zeige $T'_\alpha \subset T_\alpha$:

Sei $y \in T'_\alpha$. Falls $y \neq x$, folgt sofort $y \in T_\alpha$.

Falls nun $y = x \in T'_\alpha$, gilt:

$\alpha < t'(x) + \tau(x)$ hieraus folgt mit $\alpha \geq t(x)$:

$$t(x) \leq \alpha < t'(x) + \tau(x) < t(x) + \tau(x) \text{ womit } x \in T_\alpha \text{ folgt.}$$

Somit gilt in diesem Fall für Nebenbedingung (c):

$$\sum_{x \in T'_\alpha} \mu(z) \leq \sum_{x \in T_\alpha} \mu(z) \leq s(\alpha)$$

Falls $\alpha < t'(x)$, so folgt:

$$\forall z \in T'_\alpha : z \neq x \Rightarrow \forall z \in T'_\alpha : t'(z) = t(z) \Rightarrow \forall z \in T'_\alpha : z \in T_\alpha.$$

Somit gilt $T'_\alpha \subset T_\alpha$ und damit auch (c).

Falls $t'(x) \leq \alpha < t(x)$: Zeige: $T'_\alpha \subset T_{t(x)}$

Sei $y \in T'_\alpha$.

$$1. \text{ Fall: } y = x: \quad t(x) \leq t(x) < t(x) + \tau(x) \Rightarrow x \in T_{t(x)}$$

$$2. \text{ Fall: } y \neq x: \quad t(y) = t'(y) \leq \alpha < t'(y) + \tau(y) = t(y) + \tau(y)$$

Ann.: $t(x) \geq t'(y) + \tau(y)$ Der Gleichheitsfall kann nicht eintreten wegen der Hauptannahme des Beweises. Somit gilt: $t(y) + \tau(y) < t(x)$. Damit ist y ein Element von $T_{<}$. Somit gilt:

$$t(y) + \tau(y) \leq \max \{ t(z) + \tau(z) : z \in T_{<} \} \leq t'(x) \leq \alpha < t'(y) + \tau(y) = t(y) + \tau(y)$$

Dies ist jedoch ein Widerspruch. Damit gilt: $t(x) < t(y) + \tau(y)$

mit $\alpha < t(x)$ folgt:

$t(y) = t'(y) \leq \alpha < t(x) < t(y) + \tau(y)$, hieraus folgt sofort: $y \in T_{t(x)}$

Somit gilt insgesamt (mit s konstant):

$$\sum_{z \in T'_\alpha} \mu(z) \leq \sum_{z \in T_{t(x)}} \mu(z) \leq s(t(x)) = s(\alpha)$$

zu(d): Sei $z \in T : t'(z) \leq t(z) \leq t_{\max}$.

zu(e): Sei $z \in T$: Falls $z \neq x$, gilt: $t'(z) = t(z) \geq \text{rel}(z)$. Falls $z = x$, gilt:

$$t'(z) = t'(x) = \max \{\alpha^+, \text{rel}(x)\} \geq \text{rel}(x) = \text{rel}(z)$$

zu(f): Sei $z \in T$. Mit $(+)$ folgt: $t'(z) \leq t(z) \leq \text{due}(z)$.

Damit ist t' eine zulässige Lösung von GRCS.

Zeige nun (i) für das Element x : $t'(x) = \max \{\alpha^+, \text{rel}(x)\}$. Falls $t'(x) = \text{rel}(x)$, ist nichts zu zeigen. Sei nun $t'(x) = \alpha^+$. Da $T < \cup V(x) \cup N(x)$ eine endliche Menge ist, gibt es ein y aus $T < \cup V(x) \cup N(x)$, sodaß das Maximum bei der entsprechenden Funktion angenommen wird. Dieses y leistet das Gewünschte.

Zeige nun $(\#)$: Da gilt: $\forall y \in T: t'(y) \leq t(y)$ folgt sofort $(\#)$.

Als letztes muß noch gezeigt werden, daß das oben beschriebene Updating von t in endlich vielen Schritten die gewünschte zulässige Lösung liefert. Dazu genügt es, daß gilt:

$$|T| \leq \sum_{x \in T} t'(x) < \sum_{x \in T} t(x)$$

Denn bei jedem Updating wird die Summe um mindestens 1 verringert.
qed.

Mit Satz 13 kann eine Normalform für zulässige Lösungen von GRCS erreicht werden, sobald die Gesamtverfügbarkeiten der Ressourcen konstant sind.

6. Schranken für GRCS

In diesem Kapitel werden einige Schranken für GRCS bewiesen. Diese Schranken sind für UTS schon seit langem bekannt. Hier wird die Verallgemeinerung auf GRCS vorgenommen. Diese Schranken dienen bei der algorithmischen Lösung von RCS zur Steuerung des "branch and bound" - Verfahrens. Als erstes folgt eine Verallgemeinerung einer trivialen Schranke für UTS. Wie bei den vorangegangenen Betrachtungen stellt man auch hier fest, daß der Übergang von UTS zu GRCS nicht trivial ist. Die Schranke für UTS lautet folgendermaßen:

$$\max \{t(x) + 1 : x \in T\} \geq \lceil n/s \rceil + 1, \text{ wobei } n := \sum_{x \in T} t(x) \text{ und } \lceil n/s \rceil := \min \{m \in \mathbb{Z} : m \geq n/s\}$$

Diese Schranke besagt: Wenn in höchstens einem Zeitpunkt der Planung nicht alle Prozessoren arbeiten, so ist die entstehende Planung optimal. Für UTS ist nur eine Resource verfügbar. Im Fall von GRCS wird diese Idee auf alle Ressourcen übertragen.

1. Lemma.

Gegeben sei GRCS sowie $\alpha^* = \min \{rel(x) : x \in T\}$. Für jede Resource $\phi_l \in \Phi$ wird definiert:

$$\alpha_l := \min \left\{ \alpha \in \mathbb{N} : \sum_{j=\alpha^*}^{\alpha} s(j)_l \geq \sum_{x \in T} p(x)_l * t(x) \right\}$$

Dann gilt für jede zulässige Lösung $t : T \rightarrow \mathbb{N}$:

$$\max \{t(x) + \tau(x) : x \in T\} \geq \max \{\alpha_l : l = 1, \dots, k\}$$

Beweis:

Es sei $t : T \rightarrow \mathbb{N}$ eine zulässige Lösung von GRCS und sei

$\alpha_{\max} := \max \{t(x) + \tau(x) : x \in T\}$ und es sei ϕ_l eine beliebige Resource.

Dann gilt:

$$\sum_{x \in T} \tau(x) * \mu(x)_l = \sum_{x \in T} \sum_{\alpha = t(x)}^{t(x) + \tau(x) - 1} \mu(x)_l = \sum_{x \in T} \sum_{\substack{\alpha = \alpha^* \\ x \in T_{\alpha}}}^{\alpha_{\max}} \mu(x)_l = \sum_{\alpha = \alpha^*}^{\alpha_{\max}} \sum_{x \in T_{\alpha}} \mu(x)_l \leq \sum_{\alpha = \alpha^*}^{\alpha_{\max}} s(x)_l$$

Damit gilt:

$$\alpha_{\max} \geq \min \{ \alpha \in \mathbb{N} : \sum_{j=\alpha^*}^{\alpha} s(j)_l \geq \sum_{x \in T} \mu(x)_l * \tau(x) \} = \alpha_l$$

Da die Resource beliebig gewählt worden war, gilt:

$$\max \{ t(x) + \tau(x) : x \in T \} = \alpha_{\max} \geq \max \{ \alpha_l : l = 1, \dots, k \}. \quad \text{qed.}$$

Unmittelbar aus diesem Lemma folgt:

Falls für eine zulässige Lösung die Gleichheit gilt; d.h. $\max \{ t(x) + \tau(x) : x \in T \} = \max \{ \alpha_l : l = 1, \dots, k \}$, so ist sie schon optimal.

Auf den gerichteten Graphen $(T^*, \text{sym}(R^*))$ aus (5.3) mit $T^* := T \cup \{x^*\}$ und $R^* := R \cup (\{x^*\} \times T)$ wird nun eine Pfeilbewertung $c : R^* \rightarrow \mathbb{Z}$ definiert:

$$c(x, y) := \begin{cases} \text{rel}(y) & \text{für } x = x^* \\ \lambda(x, y) & \text{für } (x, y) \in R \\ -\kappa(y, x) & \text{für } (x, y) \in d(R) \\ -\text{due}(x) & \text{für } y = x^* \end{cases}$$

Es werden nun Ketten in $(T^*, \text{sym}(R^*))$ betrachtet. Es folgt nun ein wichtiges Lemma für die Bereitstellung von Schranken für GRCS:

2. Lemma.

Gegeben sei GRCS und $t : T \rightarrow \mathbb{N}$ zulässig für GRCS. Es sei (K, f) eine längste Kette in $(T^*, \text{sym}(R^*))$ mit Anfangspunkt in T . Dann gilt:

- (i) $f(K) \in T$

$$(ii) \quad l_c(K, f) \leq t(f(K/)) - t(f(1))$$

Beweis:

zu (i): Ann.: $f(K/) = x^*$.

Definiere $z := f(K/-1)$. Die Länge von (K, f) könnte durch Weglassen von x^* um $c(z, x^*)$ verkürzt werden. Da aber $c(z, x^*) = -\text{due}(z) < 0$, folgt daß die Verkürzung über einen Pfeil mit negativer Bewertung erfolgt, was einer echten Vergrößerung der Länge entspricht. Dies ist ein Widerspruch zur Voraussetzung, daß (K, f) eine längste Kette war.

zu (ii): Sei $x := f(1)$ und $y := f(K/)$.

1. Fall: $x^* \in K$:

$$n := f^{-1}(x^*)$$

$$R_{K,f} := \{ (f(j), f(j+1)) : j = 1, \dots, n-2, n+1, \dots, /K/ \}$$

$$l_c(K, f) = \sum_{(u,v) \in R_{K,f}} c(u, v) + c(f(n-1), x^*) + c(x^*, f(n+1)) =$$

$$= \sum_{(u,v) \in R_{K,f} \cap R} c(u, v) + \sum_{(u,v) \in R_{K,f} \cap d(R)} c(u, v) - \text{due}(f(n-1)) + \text{rel}(f(n+1)) \geq$$

$$\geq \sum_{(u,v) \in R_{K,f} \cap R} \lambda(u, v) + \sum_{(u,v) \in R_{K,f} \cap d(R)} -\kappa(v, u) - t(f(n-1)) + t(f(n+1)) \geq$$

$$\geq \sum_{(u,v) \in R_{K,f} \cap R} t(v) - t(u) - \sum_{(u,v) \in R_{K,f} \cap d(R)} t(v) - t(u) - t(f(n-1)) + t(f(n+1)) =$$

$$= \sum_{j=1}^{n-2} t(f(j+1)) - t(f(j)) + \sum_{j=n+1}^{/K/-1} t(f(j+1)) - t(f(j)) - t(f(n-1)) + t(f(n+1)) =$$

$$= t(f(n-1)) - t(f(1)) + t(f(/K/)) - t(f(n+1)) - t(f(n-1)) + t(f(n+1)) = t(y) - t(x)$$

2. Fall:

$$x^* \notin K:$$

$$R_{K,f} := \{ (f(j), f(j+1)) : j = 1, \dots, K-1 \}$$

$$l_c(K, f) =$$

$$\sum_{(u,v) \in R_{K,f}} c(u,v) = \sum_{(u,v) \in R_{K,f} \cap R} c(u,v) + \sum_{(u,v) \in R_{K,f} \cap d(R)} c(u,v) = \sum_{(u,v) \in R_{K,f} \cap R} \lambda(u,v) + \sum_{(u,v) \in R_{K,f} \cap d(R)} -\kappa(v,u) \geq$$

$$\sum_{(u,v) \in R_{K,f} \cap R} t(v) - t(u) - \sum_{(u,v) \in R_{K,f} \cap d(R)} t(v) - t(u) = \sum_{(u,v) \in R_{K,f}} t(v) - t(u) = t(f(K)) - t(f(1)) = t(y) - t(x)$$

qed.

3. Folgerung.

Gegeben sei GRCS und $t : T \longrightarrow \mathbb{N}$ zulässig für GRCS. Definiere eine Abbildung $h : T \longrightarrow \mathbb{Z}$ mit

$$h(x) := \max \{ l_c(K, f) + \tau(f(K)) : (K, f) \text{ Kette in } \text{sym}(R^*) \text{ mit } x = f(1) \in T \}$$

Dann gilt:

$$\forall x \in T : \quad t(x) + h(x) \leq \max \{ t(z) + \tau(z) : z \in T \}$$

Beweis:

Sei $x \in T$ und (K, f) eine Kette in $\text{sym}(R^*)$ mit:

- $f(1) = x$
- $f(K) =: y \in T$ (* nach (2)(i) *)
- $l_c(K, f) + \tau(y) = h(x)$

Dann gilt mit (2)(ii):

$$t(x) + h(x) = t(x) + l_c(K, f) + \tau(y) \leq t(x) + t(y) - t(x) + \tau(y) \leq \max \{ t(z) + \tau(z) : z \in T \}$$

qed.

4. Folgerung.

Sei $t^* : T \longrightarrow \mathbb{N}$ eine Optimallösung von GRCS. Dann gilt (mit den Bezeichnungen

gen von (3)):

$$\max \{h(z) + \text{rel}(z) : z \in T\} \leq \max \{t^*(z) + \tau(z) : z \in T\}$$

Beweis:

$$\text{Sei } x_0 \in T \text{ mit } h(x_0) + \text{rel}(x_0) = \max \{h(z) + \text{rel}(z) : z \in T\}.$$

Dann gilt:

$$\max \{h(z) + \text{rel}(z) : z \in T\} = h(x_0) + \text{rel}(x_0) \leq h(x_0) + t^*(x_0) \leq \max \{t^*(z) + \tau(z) : z \in T\}$$

qed.

Falls für eine zulässige Lösung von GRCS in (4) die Gleichheit gilt, so ist sie schon eine Optimallösung von GRCS.

Das folgende Lemma zeigt eine von /HU-61/ (für UTS) entwickelte Schranke nun für GRCS. Diese Schranke entsteht durch Kombination von Lemma (1) und (4).

5. Lemma.

Gegeben sei GRCS und eine zulässige Lösung $t: T \rightarrow \mathbb{N}$ sowie

$\alpha^* := \min \{\text{rel}(x) : x \in T\}$. Definiere eine Abbildung $h: T \rightarrow \mathbb{Z}$ wie in (3).

Sei nun $j \in \mathbb{N}$: $T_j := \{x \in T : \exists y \in N_{t(R)}(x) : h(y) \geq j\} \neq \emptyset$

$$\alpha_{1j} := \min \left\{ \alpha \in \mathbb{N} : \sum_{i=\alpha^*}^{\alpha} s(i)_1 \geq \sum_{x \in T_j} \mu(x)_1 * \tau(x) \right\}$$

Dann gilt: $\forall l = 1, \dots, k: \quad j + \alpha_{1j} \leq \max \{t(x) + \tau(x) : x \in T\}$

Beweis:

Zeige zuerst: $\forall x \in T_j: \quad j + t(x) + \tau(x) \leq \max \{t(z) + \tau(z) : z \in T\}.$

T_j enthält all diejenigen Elemente von T , für die es eine Kette in R zu einem Element $y \neq x$ gibt, welches einen h -Wert von mindestens j aufweist.

Begründung:

Sei $x \in T_j$. Nach Definition von T_j gibt ein $y \in T$ und eine Kette (K_1, f_1) von x nach y

in R , sodaß $h(y) \geq j$. Wähle (K_2, f_2) in (T, R) mit

- $f_2(1) = y$
- $f_2(K) = : z \in T$ (* nach (2)(i) *)
- $h(y) = l_c(K_2, f_2) + \tau(z)$

Da (K_1, f_1) eine Kette in (T, R) von x nach y ist, gilt:

$$t(y) \geq t(f_1(2)) \geq t(x) + \lambda(x, f_1(2)) \geq t(x) + \tau(x)$$

(K_2, f_2) ist eine längste Kette in $\text{sym}(R^*)$ mit $f_2(1) = y$. Damit folgt mit (2)(ii):

$$t(y) \leq t(z) - l_c(K_2, f_2)$$

Damit gilt insgesamt:

$$t(x) + \tau(x) + j \leq t(y) + j \leq t(z) - l_c(K_2, f_2) + j \leq t(z) - l_c(K_2, f_2) + h(y) =$$

$$t(z) - l_c(K_2, f_2) + l_c(K_2, f_2) + \tau(z) = t(z) + \tau(z) \leq \max \{ t(w) + \tau(w) : w \in T \}$$

Nach (1) gilt:

$$\forall l = 1, \dots, k : \alpha_{lj} \leq \max \{ t(x) + \tau(x) : x \in T_j \} = t(z_0) + \tau(z_0) \text{ mit } z_0 \in T_j.$$

Damit gilt insgesamt:

$$\forall l = 1, \dots, k :$$

$$j + \alpha_{lj} \leq j + \max \{ t(x) + \tau(x) : x \in T_j \} = j + t(z_0) + \tau(z_0) \leq \max \{ t(x) + \tau(x) : x \in T \}$$

qed.

6. Folgerung.

Ummittelbar aus Lemma 5 folgt:

- (i) $\max \{ j + \alpha_{lj} : j = 1, \dots, t_{\max} T_j \neq \emptyset, \phi_l \in \Phi : l = 1, \dots, k \} \leq \max \{ t(x) + \tau(x) : x \in T \}$
- (ii) Falls in Lemma 5 die Gleichheit gilt, ist die zulässige Lösung t schon eine Optimallösung von GRCS.

Im folgenden wird eine duale Aussage zu Lemma 5 bewiesen. Dazu betrachtet man auf $\text{sym}(R^*)$ folgende geänderte Kantenbewertung $c : \text{sym}(R^*) \longrightarrow \mathbb{Z}$

$$c(x,y) := \begin{cases} -\text{due}(y) & \text{für } x = x^* \\ -\kappa(x,y) & \text{für } (x,y) \in R \\ \lambda(y,x) & \text{für } (x,y) \in d(R) \\ \text{rel}(x) & \text{für } y = x^* \end{cases}$$

7. Lemma.

Gegeben sei GRCS zusammen mit einer zulässigen Lösung $t : T \longrightarrow \mathbb{N}$. Dann gilt:

$$\forall x \in T : t(x) \geq h(x),$$

wobei die Abbildung $h : T \longrightarrow \mathbb{Z}$ folgendermaßen definiert ist:

$$h(x) := \max \{ l_c(K,f) : (K,f) \text{ Kette in } \text{sym}(R^*) \text{ mit } f(1) = x \}$$

Beweis:

Sei $x \in T$ beliebig. Wähle eine Kette (K,f) in $\text{sym}(R)$ mit

$$f(1) = x \text{ und } h(x) = l_c(K,f)$$

Es gilt $x^* \in K$, denn sonst könnte (K,f) um x^* und $l(K,f)$ um $\text{rel}(f(K)) > 0$ vermehrt werden.

$$\text{Ann.: } f^{-1}(x^*) := n \neq |K|$$

Dann hat die Teilkette von x^* nach $f(K)$ nichtnegative Bewertung. Diese Teilkette ist sogar ein Zyklus in $\text{sym}(R^*)$, da $(f(K), x^*) \in \text{sym}(R^*)$. Dieser Zyklus hat echt positive Bewertung, da $c(f(K), x^*) = \text{rel}(f(K)) > 0$.

Für die Pfeilbewertung $-c$ ist dies dann ein negativ bewerteter Zyklus. Da $-c$ gerade die Pfeilbewertung im Satz (5.2) ist, folgt mit dem Beweis von (5.2), daß es keine zulässige Lösung von GRCS gibt. - Widerspruch -

Damit gilt $n = /K/$.

$$R_{K,f} := \{ (f(j), f(j+1)) : j = 1, \dots, /K/ - 2 \}$$

$$y := f(/K/ - 1)$$

Damit ergibt sich:

$$l_c(K, f) = \sum_{(u,v) \in R_{K,f}} c(u,v) + c(y, x^*) = \sum_{(u,v) \in R_{K,f} \cap R} -\kappa(u,v) + \sum_{(u,v) \in R_{K,f} \cap d(R)} \lambda(v,u) + \text{rel}(y) \leq$$

$$\sum_{(u,v) \in R_{K,f} \cap R} t(u) - t(v) - \sum_{(u,v) \in R_{K,f} \cap d(R)} t(u) - t(v) + \text{rel}(y) = \sum_{(u,v) \in R_{K,f}} t(u) - t(v) + \text{rel}(y) =$$

$$t(x) - t(y) + \text{rel}(y) \leq t(x)$$

qed.

Nun folgt die schon angekündigte Dualaussage zu Lemma 2:

8. Lemma.

Gegeben sei GRCS und eine zulässige Lösung $t : T \longrightarrow \mathbb{N}$. Es sei $j \in \{1, \dots, t_{\max}\}$, sodaß die Menge $T^j := \{x \in T : h(x) \geq j\}$ nicht leer ist. Es sei $\phi_l \in \Phi$ eine beliebige Resource. Definiere $\alpha^* := \min \{h(x) : x \in T^j\}$ und sei

$$\beta_{l,j} := \min \left\{ \alpha \in \mathbb{N} : \sum_{i=\alpha^*}^{\alpha} s(i)_l \geq \sum_{x \in T^j} \mu(x)_l * \tau(x) \right\}$$

$$\text{Dann gilt: } \beta_{l,j} \leq \max \{ t(x) + \tau(x) : x \in T \}$$

Beweis:

Betrachte folgendes Unterproblem $\text{GRCS}' := (T^j, R(T^j), \tau|_{T^j}, \mu|_{T^j}, s, \text{rel}', \text{due}, t_{\max})$ mit folgender Abbildung $\text{rel}' : T^j \longrightarrow \mathbb{N}$ mit $\text{rel}'(x) := h(x)$.

Das Problem GRCS' hat eine Lösung, denn für $t|_{T^j}$ gilt mit Lemma 7:

$$\forall x \in T^j : t(x) \geq h(x) = \text{rel}'(x)$$

Wende nun Lemma 1 auf GRCS' an. Es gilt mit $\alpha^* = \min \{ \text{rel}'(x) : x \in T^j \}$:

$$\max \{ t(x) + v(x) : x \in T \} \geq \max \{ t(x) + v(x) : x \in T^j \} \geq (* \text{ Lemma 1} *)$$

$$\min \{ \alpha \in \mathbb{N} : \sum_{i=\alpha^*}^{\alpha} s(i)_l \geq \sum_{x \in T^j} u(x)_l * v(x) \} = \beta_{l,j}$$

Dies beendet den Beweis.

Da der Zeitpunkt j sowie die Resource $\phi_l \in \Phi$ beliebig waren, ergibt sich als unmittelbare Folgerung:

$$(i) \max \{ t(x) + v(x) : x \in T \} \geq \max \{ \beta_{l,j} : j \in \{1, \dots, t_{\max}\} : T^j \neq \emptyset \text{ und } \phi_l \in \Phi : l = 1, \dots, k \}$$

(ii) Falls in (i) die Gleichheit gilt, ist die zulässige Lösung t schon eine Optimallösung von GRCS.

7. Anordnungsprobleme

In diesem Kapitel werden Anordnungsprobleme formuliert. Während die Lösung von Schedulingaufgaben die Anfangszeitpunkte für die Elementaroperationen aus T bereitstellt, liefert die Lösung des Anordnungsproblems den Ort (d.h. den spezifischen Prozessor), auf dem die einzelnen Elementaroperationen ablaufen. Anordnungsprobleme weisen Ähnlichkeiten zu Registerzuordnungsproblemen (allocation of registers) im Compilerbau auf /AHO-79/.

Beim Entwurf von Rechnerarchitekturen (synchronous systems) wurden von /QUINTON-84/ Anordnungsprobleme in ähnlicher Weise formuliert. Dort wird beim systematischen Entwurf von systolischen Systemen das Gesamtproblem in ein Scheduling- und ein Anordnungsproblem zerlegt.

Zunächst sei an die Bezeichnungen aus (4.3) erinnert:

T	Menge der Elementaroperationen
$R \subset v(T)$	zeitliches Nacheinander der Elementaroperationen
$\tau : T \longrightarrow \mathbb{N}$	Länge der Ausführungsintervalls
$t_{\max}$	maximaler Ausführungsbeginn
$\Phi := \{\phi_1, \dots, \phi_k\}$	Menge der Ressourcen
$\mu : T \longrightarrow \mathbb{N}_0^k$	Ressourcenanforderungen
$s \in \mathbb{N}_0^k$	Gesamtverfügbarkeiten

1. Prozessoren.

Prozessoren sind Objekte, auf denen Elementaroperationen ausgeführt werden können. Sie stellen eine hardwaremäßige Implementierung von Elementaroperationen dar. Jeder Prozessor verfügt über eine Menge von Ressourcen. Diese Ressourcen stellen die Implementierungsmöglichkeiten des einzelnen Prozessors dar. Damit ist ein Prozessor genau dann eine Implementierung einer gewissen Elementaroperation, wenn er über die Ressourcen verfügt, die die Elementaroperation anfordert.

Ein weiterer charakterisierender Parameter eines Prozessors ist seine Geschwindigkeit $\delta \in]0, 1]$. Der Zeitbedarf einer Implementierung der Elementaroperation x auf diesem Prozessor ergibt sich aus den Produkt $\delta * \tau(x)$.

2. Datenleitungen.

Die azyklische Relation R beschreibt das zeitliche Nacheinander der einzelnen Elementaroperationen. Ein wichtiger (wenn auch nicht ausschließlicher) Grund für dieses Nacheinander ist die Datenabhängigkeit zweier Elementaroperationen /MIRANKER-84/, /MOLDOVAN-83/, /QUINTON-84/, /RAMAKRISH-84/. Eine Elementaroperation benötigt Daten, die von anderen erst bereitgestellt bzw. berechnet werden. Außer der Datenabhängigkeit gibt es noch andere Gründe, meist planungstechnischer Natur (z.B. periodisches Zeitverhalten), die ein zeitliches Nacheinander erfordern. Falls Sende- und Empfangsoperation auf verschiedenen Prozessoren implementiert sind, müssen für die Datenabhängigkeiten Verbindungsleitungen zwischen ihnen bereitgestellt werden. Diese Verbindungsleitungen werden als Pfeile eines gerichteter Graphen auf der Menge der Prozessoren modelliert. Hierbei spielt das Datenformat (z.B. Anzahl der bits) und das Übermittlungsprotokoll (Organisation des Datenaustauschs) noch keine Rolle. Ein wichtiges Charakteristikum einer Leitung ist der *Verzögerungswert*. Er gibt an, nach wievielen Zeittakten das Eingangssignal einer Leitung am Ausgang verfügbar ist. Da in dieser Arbeit nur synchrone Systeme betrachtet werden, ist der Verzögerungswert jeder Leitung stets nichtnegativ. Ein Verzögerungswert w gibt an, daß das Eingangssignal zu Beginn des $w + 1$ -nächsten Zeittaktes am Ausgang verfügbar ist. Somit ist die Signalausbreitung innerhalb eines Zeittaktes über eine Datenleitung ausgeschlossen (kein combinatorial rippling /LEISERSON-83/). In (vorläufigen) Entwürfen synchroner Systeme ist für manche Leitungen nur eine Obergrenze w des Verzögerungswerts gegeben.

Zur formalen Beschreibung von (1) und (2) werden nun zusätzlich folgende Bezeichnungen gewählt:

3. Bezeichnungen.

P bezeichne die endliche Menge der Prozessoren. Die verfügbaren Ressourcen der Prozessoren gibt die Abbildung $\eta : P \longrightarrow \mathbb{N}_0^k$ an. Die Geschwindigkeit der Prozessoren beschreibt die Abbildung $\delta : P \longrightarrow]0, 1]$.

Die Teilmenge $R_d \subset R$ gibt die Datenabhängigkeiten der Elementaroperationen aus T an. Der gerichtete Graph (P, W) mit $W \subset v(P)$ gibt die logischen Verbindungen (Leitungen) zwischen den Prozessoren an. Die Menge W zerfällt in zwei Teil-

mengen W_l und W_f . Die Menge $W_f(W_l)$ beinhaltet die Leitungen mit festem (limitiertem) Verzögerungswert. Die Abbildung $\xi: W \longrightarrow N_0$ gibt für $x \in W_f$ den Verzögerungswert und für $x \in W_l$ die Obergrenze desselben an.

Mit den oben gewählten Bezeichnungen können nun Anordnungsprobleme formuliert werden. Dazu müssen noch einige Begriffe eingeführt werden:

Gegeben sei $T, R, \tau, \Phi, \mu, s, t_{\max}$, die Bezeichnungen aus (3) und eine Abbildung $t: T \longrightarrow N$ (Anfangszeitpunkte) mit $\forall x \in T: t(x) \leq t_{\max}$.

Auf der Menge T werden nun $/P/$ ungerichtete Graphen $I_p \subset v(T): p \in P$ definiert:

$$(x, y) \in I_p: \Leftrightarrow \begin{cases} t(y) < t(x) + \tau(x) * \delta(p) & \text{falls } t(x) \leq t(y) \\ t(x) < t(y) + \tau(y) * \delta(p) & \text{falls } t(x) > t(y) \end{cases}$$

Die Graphen $I_p: p \in P$ definieren Inkompatibilitäten zwischen Elementaroperationen. Zwei Elementaroperationen x, y sind bez. des Prozessors p inkompatibel, wenn sich die Ausführungsintervalle der Implementierung auf p überlappen:

$$[t(x), t(x) + \tau(x) * \delta(p)] \cap [t(y), t(y) + \tau(y) * \delta(p)] \neq \emptyset$$

Damit sind die $I_p: p \in P$ Intervallgraphen /GOLUMBIC-80/ auf der Menge T .

Definiere nun eine Relation $E \subset T \times P$:

$$(x, p) \in E: \Leftrightarrow \begin{cases} \text{(i)} & \mu(x) \leq \eta(p) \leq s \\ \text{(ii)} & t(x) + \tau(x) * \delta(p) \leq \min \{ t(y) : y \in N_R(x) \} \end{cases}$$

Ein Paar (x, p) ist demnach genau dann in E , wenn Prozessor p alle Ressourcen für x bereitstellt (unter Beachtung der Gesamtrestriktion) und wenn das Ausführungsende der Implementierung von x auf p so früh erreicht wird, daß alle Nachfolger von x in R erst später ihre Ausführung beginnen. Die Begriffe Vorgänger bzw. Nachfolger werden auch auf E in intuitiver Weise benützt.

Definiere zuletzt für alle $p \in P$ eine Abbildung $r_p: R_d \longrightarrow Z$ mit:

$$r_p(x, y) := \lfloor t(y) - (t(x) + \delta(p) * \tau(x)) \rfloor$$

Hierbei bezeichnet $\lfloor x \rfloor$ mit $x \in \mathbb{R}$ die größte ganze Zahl kleiner-gleich x . Die Abbildung $r_p, p \in P$ gibt den Verzögerungswert für die *Datenabhängigkeit* $(x, y) \in R_d$ für eine Implementierung von x auf p an. Falls t zulässig für (RCS) ist, ist r_p stets nichtnegativ, da $\delta \leq 1$.

Weiters wird eine Projektionsabbildung $\text{pr}_p : E \longrightarrow P$ definiert durch:

$$\text{pr}_p(x, p) := p$$

4. Definition. (GAP - "general allocation problem")

Gegeben seien $T, P, R, R_d, (I_p)_{p \in P}, W, W_1, W_f, (r_p)_{p \in P}, \xi, \Phi, \eta, s, \text{pr}_p$.

Gegeben sei weiterhin eine Kostenbewertung $c : E \longrightarrow \mathbb{R}$.

Gesucht wird eine Abbildung $a : E \longrightarrow \{0, 1\}$, sodaß

$$cTa := \sum_{(x,p) \in E} c(x,p) * a(x,p) \longrightarrow \min$$

unter den Nebenbedingungen:

$$(a) \quad \forall x \in T \quad \sum_{p \in N_E(x)} a(x,p) = 1$$

$$(b) \quad \forall p \in P \quad \forall (x,y) \in I_p : \quad a(x,p) + a(y,p) \leq 1$$

$$(c) \quad \forall (x,y) \in R_d$$

$$(1) \quad \forall (p,q) \in v(P) \setminus W : (x,p), (y,q) \in E \Rightarrow a(x,p) + a(y,q) \leq 1$$

$$(2) \quad \forall (p,q) \in W_f : (x,p), (y,q) \in E \wedge \xi(p,q) \neq r_p(x,y) \Rightarrow a(x,p) + a(y,q) \leq 1$$

$$(3) \quad \forall (p,q) \in W_1 : (x,p), (y,q) \in E \wedge \xi(p,q) < r_p(x,y) \Rightarrow a(x,p) + a(y,q) \leq 1$$

$$(d) \quad \sum_{p \in \text{pr}_p(a^{-1}(1))} \eta(p) \leq s$$

enddef.

Die Abbildung $a : E \longrightarrow \{0, 1\}$ stellt eine Entscheidungsfunktion dar. Die Menge E repräsentiert alle möglichen Implementierungen der Operationen aus T auf den Prozessoren P . Die Abbildung a wählt nun aus den möglichen die vorzunehm-

menden Zuordnungen aus. Eine Elementaroperation x wird nämlich genau dann auf Prozessor p implementiert, wenn $a(x,p) = 1$ ist.

Die Nebenbedingungen repräsentieren folgende Einschränkungen:

- (a): Jede Elementaroperation muß genau einem Prozessor zugeordnet werden.
- (b): Falls zwei Elementaroperationen x, y bez. Prozessor p inkompatibel sind (d.h. $(x, y) \in I_p$) und sowohl x als auch y dem Prozessor p zugeordnet werden können (d.h. $(x, p), (y, p) \in E$), dann dürfen nicht beide Operationen dem Prozessor p zugeordnet werden (d.h. nicht $a(x, p) = 1$ und $a(y, p) = 1$).
- (c): Falls eine Elementaroperation y von einer anderen Operation x datenabhängig ist (d.h. $(x, y) \in R_d$) und x dem Prozessor p bzw. y dem Prozessor q zugeordnet werden kann (d.h. $(x, p), (y, q) \in E$), dann muß eine Leitung von p nach q vorhanden sein. Dies wird dadurch ausgedrückt, daß für alle nicht vorhandenen Leitungen $(p', q') \in v(P) \setminus W$ nicht $a(x, p')$ und $a(y, q')$ identisch eins sein dürfen. Auch darf (x, y) nicht auf eine Datenleitung $(p', q') \in W_f$ abgebildet werden, deren Verzögerungswert $\xi(p', q') \neq r_p(x, y)$. Ebenso scheidet die Datenleitung $(p', q') \in W_l$ aus, für die gilt: $\xi(p', q') < r_p(x, y)$.
- (d): Die Menge $a^{-1}(1)$ wird durch pr_p auf P projiziert. Die Menge $pr_p(a^{-1}(1))$ enthält diejenigen Prozessoren, die als Zuordnungsziel für irgendeine Elementaroperation dienen, denn es gilt:

$$\forall p \in P: \quad p \in pr_p(a^{-1}(1)) \Leftrightarrow \exists x \in T: a(x, p) = 1$$

Damit besagt Nebenbedingung (d), daß die zur Verfügung gestellten Ressourcen sämtlicher benützter Prozessoren die Gesamtverfügbarkeiten an Resourcen nicht übersteigen darf.

Die Frage der Existenz von Lösungen wird bei der Entwicklung eines algorithmischen Verfahrens im Mittelpunkt stehen, denn man ist in der Regel mit einer zulässigen Lösung von GAP zufrieden.

Es werden nun noch einfachere Varianten von GAP betrachtet: Zuerst werden die Gesamtressourcen s weggelassen. Damit ergibt sich ein lineares ganzzahliges Programm:

5. Lineares Anordnungsproblem (LAP - "linear allocation problem")

Gegeben seien die Bezeichnungen aus (4).

Es sei $c : E \longrightarrow \mathbb{R}$ eine beliebige Bewertung auf E .

Gesucht wird eine Abbildung $a : E \longrightarrow \{0,1\}$, sodaß:

$$c^T a \longrightarrow \min$$

unter den Nebenbedingungen:

$$(a) \quad \forall x \in T \quad \sum_{p \in N_E(x)} a(x,p) = 1$$

$$(b) \quad \forall p \in P \quad \forall (x,y) \in I_p : \quad a(x,p) + a(y,p) \leq 1$$

$$(c) \quad \forall (x,y) \in R_d$$

$$(1) \quad \forall (p,q) \in v(P) \setminus W : \quad (x,p), (y,q) \in E \quad \Rightarrow \quad a(x,p) + a(y,q) \leq 1$$

$$(2) \quad \forall (p,q) \in W_f : \quad (x,p), (y,q) \in E \wedge \xi(p,q) \neq r_p(x,y) \Rightarrow a(x,p) + a(y,q) \leq 1$$

$$(3) \quad \forall (p,q) \in W_l : \quad (x,p), (y,q) \in E \wedge \xi(p,q) < r_p(x,y) \Rightarrow a(x,p) + a(y,q) \leq 1$$

$$(d) \quad -$$

enddef.

LAP ist ein ganzzahliges lineares Programm, bei dem die Gesamtverfügbarkeiten der Ressourcen nicht mehr beachtet werden.

Nun folgt die kontinuierliche Version LAP_{con} von LAP:

6. Kontinuierliches lineares Anordnungsproblem LAP_{con}

Gegeben seien die Bezeichnungen aus (5).

Gesucht wird eine Abbildung $a : E \longrightarrow \mathbb{R}$, sodaß:

$$c^T a \longrightarrow \min$$

unter den Nebenbedingungen:

$$(a) \quad \forall x \in T \quad \sum_{p \in N_E(x)} a(x,p) = 1$$

- (b) $\forall p \in P \quad \forall (x,y) \in I_p: \quad a(x,p) + a(y,p) \leq 1$
- (c) $\forall (x,y) \in R_d$
- (1) $\forall (p,q) \in v(P) \setminus W: \quad (x,p), (y,q) \in E \quad \Rightarrow \quad a(x,p) + a(y,q) \leq 1$
 - (2) $\forall (p,q) \in W_f: \quad (x,p), (y,q) \in E \wedge \xi(p,q) \neq r_p(x,y) \quad \Rightarrow \quad a(x,p) + a(y,q) \leq 1$
 - (3) $\forall (p,q) \in W_l: \quad (x,p), (y,q) \in E \wedge \xi(p,q) < r_p(x,y) \quad \Rightarrow \quad a(x,p) + a(y,q) \leq 1$
- (d) -
- (e) $\forall (x,p) \in E: \quad a(x,p) \geq 0$

enddef.

Man kann die Abbildung $a: E \longrightarrow \mathbf{R}$ auch als Vektor $a \in \mathbf{R}^{E/}$ betrachten, wodurch die Konsistenz mit der üblichen Schreibweise der linearen Programmierung gewahrt ist. Aus Gründen der Übersichtlichkeit wird jedoch an der Abbildungsschreibweise festgehalten.

Ein sehr wichtiger Gesichtspunkt bei den folgenden Untersuchungen wird sein, in wie weit eine Lösung von LAP_{con} identisch mit einer Lösung von LAP ist.

Die Einschränkungen von UTS können auch bei Anordnungsproblemen betrachtet werden:

Alle Elementaroperationen sind identisch, haben die Ausführungszeit 1 und fordern genau eine Einheit der einzigen Resource ϕ_1 :

- | | |
|--------------------|---|
| $\tau \equiv 1$ | konstante Ausführungszeit |
| $k = 1$ | genau eine Resource |
| $s \in \mathbf{N}$ | Insgesamt höchstens s Einheiten der einen Resource |
| $\mu \equiv 1$ | Anforderung genau einer Einheit der einzigen Resource |

Zur Implementierung sind identische Prozessoren vorgesehen:

- | | |
|-------------------|--|
| $\eta \equiv 1$ | jeder Prozessor stellt genau eine Einheit der einzigen Resource bereit |
| $\delta \equiv 1$ | konstante Geschwindigkeit aller Prozessoren |

An die Datenabhängigkeiten R_d und die Prozessorverbindungen W werden keine Einschränkungen gemacht. Die Nebenbedingung (c) von (4) wird in der Formulierung des Anordnungsproblems für Einheitoperationen weggelassen.

7. Anordnungsproblem für Einheitsoperationen

(UTAP - "unit task allocation problem")

Die relevanten Bezeichnungen für UTAP sind: $T, R, s, t: T \longrightarrow \mathbb{N}$.

$E := \{ (x, p) \in T \times P : t(x) < \min \{ t(y) : y \in N_R(x) \} \}$.

Es sei P die Menge der Prozessoren und es gelte $|P| = s$.

Gegeben ist eine Kostenabbildung $c: E \longrightarrow \mathbb{R}$.

Gesucht ist eine Abbildung $a: E \longrightarrow \{0, 1\}$, sodaß

$cTa \longrightarrow \min$

unter den Nebenbedingungen:

$$(a) \quad \forall x \in T \quad \sum_{p \in N_E(x)} a(x, p) = 1$$

$$(b) \quad \forall p \in P \quad \forall x, y \in T: \quad x \neq y \wedge t(x) = t(y) \Rightarrow a(x, p) + a(y, p) \leq 1$$

(c) -

(d) -

enddef.

Das Problem UTAP ist ein lineares ganzzahliges Programm. Das nächste Kapitel zeigt, wie UTAP in ein Netzflußproblem transformiert werden kann, das mit den bekannten Algorithmen /DOMSCHKE-81/ gelöst werden kann. Die Nebenbedingung (c) (Vorhandensein von Datenleitungen) wäre jedoch der totalen Unimodularität der auf Gleichheitsrestriktionen erweiterten Nebenbedingungsmatrix im Wege gewesen. Deshalb wurde (c) in der Formulierung von UTAP weggelassen. Die Nebenbedingung (d) ist implizit in die Anzahl der Prozessoren eingeflossen. Jeder Prozessor stellt genau eine Einheit der einzigen Resource bereit. Auch jede Elementaroperation fordert genau eine Einheit dieser Resource. Somit genügt es die Prozessoren selber als einzige Resource zu betrachten. Da $s \in \mathbb{N}$ die Gesamtbeschränkung der einzigen Resource ist, genügt es die Anzahl der Prozessoren auf s zu beschränken. Damit kann die Nebenbedingung (d) entfallen.

8. Lösung von UTAP

In diesem Kapitel wird ein Existenzsatz für UTAP bewiesen. Eine Lösung von UTAP existiert genau dann, wenn die Abbildung $t: T \longrightarrow N$ eine zulässige Lösung von UTS ist. Die Lösung von UTAP erfolgt mit einem Netzflußansatz. Flußprobleme können mit bekannten Algorithmen (MoDi-Verfahren bzw. out-of-kilter-Verfahren in /DOMSCHKE-81/) gelöst werden. Auf die Beschreibung dieser Algorithmen wird hier (da schon hinreichend bekannt) verzichtet.

Vorerst werden UTS aus (4.6) und UTAP aus (7.7) in Erinnerung gerufen. Gegeben seien die Bezeichnungen aus Kapitel 7. Für "unit task" Probleme gelten folgende Einschränkungen:

$$\tau \equiv 1, \Phi = \{\phi_1\}, \mu \equiv 1, s \in N, \eta \equiv 1, \delta \equiv 1.$$

Da in der Formulierung von UTAP die Datentransfers R_d keine Rolle spielen, wird für dieses Kapitel $R_d = \emptyset$ angenommen.

Die für dieses Kapitel relevanten Bezeichnungen sind:

1. Bezeichnungen.

T	Menge der Elementaroperationen
$R \subset v(T)$	zeitliches Nacheinander der Elementaroperationen
$t_{\max}$	maximaler Ausführungsbeginn
$t: T \longrightarrow N$	Anfangszeitpunkte
$s \in N$	Gesamtressourcenbeschränkung
P	Menge der Prozessoren
$E \subset T \times P$	Mögliche Zuordnungen
	$E = \{(x, p) \in T \times P : t(x) < \min \{t(y) : y \in N_R(x)\}\}$

2. Existenzsatz.

Gegeben seien die Bezeichnungen aus (1) sowie UTS und UTAP. Es gelte weiterhin:

$$(*) \quad \forall x \in T: t(x) \leq t_{\max}$$

$$(**) \quad |P| = s$$

Dann existiert eine Optimallösung von UTAP genau dann, wenn $t : T \longrightarrow \mathbb{N}$ eine zulässige Lösung von UTS ist.

Beweis:

Die Nebenbedingungen von UTS werden mit (Sa), (Sb), (Sc), die von UTAP mit (Aa), ..., (Ad).

"nur dann": Es sei $a : E \longrightarrow \{0,1\}$ eine optimale Lösung von UTAP.

z.z.: $t : T \longrightarrow \mathbb{N}$ ist zulässig für UTS:

(Sa): $\forall (x,y) \in R : t(x) + 1 \leq t(y)$:

Sei nun $(x,y) \in R$. Nach (Aa) gilt: $\exists p \in P : a(x,p) = 1 \Rightarrow$

$(x,p) \in E \Rightarrow t(x) < \min \{ t(z) : z \in N_R(x) \} \Rightarrow t(x) < t(y)$, da $y \in N_R(x)$.

(Sb): $\forall \alpha = 1, \dots, t_{\max} : /t^{-1}(\alpha)/ \leq s$

Sei $\alpha \in \{1, \dots, t_{\max}\}$. Definiere $P_\alpha := \{p \in P : \exists x \in t^{-1}(\alpha) : a(x,p) = 1\}$

Zeige nun $/t^{-1}(\alpha)/ \leq /P_\alpha/$

Ann.: $/t^{-1}(\alpha)/ > /P_\alpha/$. Nach (Aa) gilt: $\forall x \in t^{-1}(\alpha) \exists p \in P : a(x,p) \geq 1$.

Da $/t^{-1}(\alpha)/ > /P_\alpha/$ gibt es $x,y \in t^{-1}(\alpha) : \exists p \in P : a(x,p)$ und $a(y,p) \geq 1$.

Hieraus folgt: $t(x) = \alpha = t(y)$ und $a(x,p) + a(y,p) \geq 2$.

Die ist ein Widerspruch zu (Ab).

Insgesamt :

$/t^{-1}(\alpha)/ \leq /P_\alpha/ \leq /P/ = s$ Die letzte Ungleichung gilt nach (**).

(Sc): $\forall x \in T : t(x) \leq t_{\max}$ gilt nach (*)

Damit ist t eine zulässige Lösung von UTS

"immer dann":

Es sei $t : T \longrightarrow \mathbb{N}$ eine zulässige Lösung von UTS. Definiere nun eine Ordnung

$L := \{ (x,y) \in v(T) : t(x) < t(y) \}$ auf T . Nun wird der klassische Satz von Dilworth /FISHBURN-85/ p 12 benützt, der folgendes besagt.

Für die Ordnung L über T existiert eine Zerlegung $(T_i)_{i=1,\dots,n}$ der Menge T mit:

(i) $\forall i = 1,\dots,n : L(T_i)$ ist vollständig

(ii) $n = \sigma(L)$

Somit kann L in n Ketten (lineare Teilordnungen) zerlegt werden, wobei n die Stabilitätszahl von L ist.

Die Prozessoren werden nun durchnummeriert: $P = \{p_1, \dots, p_s\}$

Damit kann die Abbildung $a : E \longrightarrow \mathbb{N}_0$ definiert werden:

$$a(x, p_i) := \begin{cases} 1 & \text{falls } x \in T_i \\ 0 & \text{sonst} \end{cases}$$

Damit die Abbildung a wohldefiniert ist, muß gezeigt werden, daß $s \geq \sigma(L)$.

Es sei $S \subset T$ eine größte L -stabile Menge in (T, L) , d.h. $|S| = \sigma(L)$.

Nach Definition von L ist t auf S konstant, d.h.

$$\exists \alpha \in \{1, \dots, t_{\max}\} : S \subset t^{-1}(\alpha)$$

Da t für UTS zulässig ist, gilt insbesondere (S_b) für α : $|t^{-1}(\alpha)| \leq s$

Damit gilt insgesamt: $\sigma(L) = |S| \leq |t^{-1}(\alpha)| \leq s$.

Behauptung: Diese Abbildung a ist für UTAP zulässig:

$$(Aa) \quad \forall x \in T : \sum_{p \in P} a(x, p) = 1$$

Sei nun $x \in T$.

Da $(T_i)_{i=1,\dots,n}$ eine Zerlegung von T ist, gibt es genau einen Index j mit

$x \in T_j$. Somit gilt: $a(x, p_j) = 1$ und $\forall i \in \{1, \dots, n\} \setminus \{j\} : a(x, p_i) = 0$, woraus

(Aa) folgt.

$$(Ab) \quad \forall p \in P \forall x, y \in T : (t(x) = t(y) \text{ und } x \neq y \Rightarrow a(x, p) + a(y, p) \leq 1)$$

Sei $p_j \in P$ und $x, y \in T$ und $x \neq y$ mit $t(x) = t(y)$ beliebig. Da $t(x) = t(y)$ ist $\{x, y\}$ stabil in L . Damit kann wegen der Vollständigkeit von $L(T_j)$ nicht x

und y in T_j liegen.

Somit gilt $a(x, p_j) = 0$ oder $a(y, p_j) = 0 \quad \Rightarrow \quad a(x, p) + a(y, p) \leq 1$.

Damit ist a für UTAP zulässig.

Nach Definition von UTAP gibt es höchstens $2^{|E|}$ zulässige Lösungen, also nur endlich viele. Somit folgt aus der Existenz der zulässigen Lösung a die Existenz einer Optimallösung von UTAP. qed.

Es soll nun die totale Unimodularität /GARFINKEL-72/ der mit Schlupfvariablen auf Gleichheitsrestriktionen gebrachten Nebenbedingungsmatrix von UTAP untersucht werden. Falls die Nebenbedingungsmatrix eines linearen ganzzahligen Programms mit Gleichheitsrestriktionen total unimodular ist, ist eine Optimallösung der kontinuierlichen Version des Programms ganzzahlig und somit auch eine Optimallösung des ganzzahligen linearen Problems (/GARFINKEL-72/ Theorem 1, p 67).

Prinzipiell können die Nebenbedingungen von UTAP so umformuliert werden, daß die resultierende Nebenbedingungsmatrix die für die totale Unimodularität hinreichende Bedingung von Theorem 3 in /GARFINKEL-72/ p 70 erfüllt. Mit diesem Wissen könnte dann UTAP mit dem Simplexalgorithmus gelöst werden. Dieser Weg wird hier jedoch nicht beschritten. Vielmehr wird ein kapazitiertes pfeilbewertetes Netzflußproblem FP /DOMSCHKE-81/ im folgenden definiert. Es wird gezeigt, daß es für UTAP eine Kostenbewertung von FP gibt, sodaß die Optimallösung von FP eine Optimallösung von UTAP induziert. Für die Lösung von Netzflußproblemen existieren dem klassischen Simplexverfahren weit überlegene spezielle Verfahren (z.B. "out-of-kilter" Algorithmus /DOMSCHKE-81/). Kapazitierte Flußprobleme sind total unimodular /GARFINKEL-72/ p 70 ff.

3. Definition. FP - "flow-problem"

Gegeben seien die Bezeichnungen aus (1) und eine Abbildung $t: T \longrightarrow \mathbb{N}$, für die gilt:

$$\forall x \in T: t(x) \leq t_{\max}.$$

Die Menge T wird partitioniert in $(T_\alpha)_{\alpha=1,\dots,t_{\max}}$ mit $T_\alpha := t^{-1}(\alpha)$. Zusätzlich zu den Mengen T und P werden noch folgende Elemente betrachtet:

v_q	Quelle des Netzwerkes
v_s	Senke des Netzwerkes
$v_{\alpha,p}$	Diese Knoten existieren für alle Zeitpunkte $\alpha = 1, \dots, t_{\max}$ und alle Prozessoren $p \in P$.

Die Knotenmenge V des Netzwerkes ist definiert durch:

$$V := \{v_q, v_s\} \cup \{v_{\alpha,p} : \alpha = 1, \dots, t_{\max} \text{ und } p \in P\} \cup T \cup P$$

Die Pfeilmenge $D \subset v(V)$ wird folgendermaßen definiert:

$$D := (\{v_q\} \times T) \cup \{(x, v_{t(x),p}) : (x,p) \in E\} \cup \\ \{(v_{\alpha,p},p) : \alpha = 1, \dots, t_{\max} \text{ und } p \in P\} \cup (P \times \{v_s\}) \cup \{(v_s, v_q)\}$$

Der minimale (maximale) Flußwert $\lambda(\kappa) : D \longrightarrow \mathbb{N}_0$ wird nun definiert:

$$\lambda(w,z) := \begin{cases} 1 & \text{für } (w,z) \in \{v_q\} \times T \\ 0 & \text{sonst} \end{cases}$$

$$\kappa(w,z) := \begin{cases} \infty & \text{für } (w,z) \in (P \times \{v_s\}) \cup \{(v_s, v_q)\} \\ 1 & \text{sonst} \end{cases}$$

Gegeben ist weiterhin eine Kostenbewertung $c_F : D \longrightarrow \mathbb{R}$

Gesucht ist eine Abbildung $\psi : D \longrightarrow \mathbb{R}$, sodaß

$$c_F^T \psi \longrightarrow \min$$

unter den Nebenbedingungen:

$$(a) \quad \forall z \in V: \quad - \sum_{w \in V_D(z)} \psi(w,z) + \sum_{w \in N_D(z)} \psi(z,w) = 0$$

$$(b) \quad \forall (z,w) \in D: \quad -\psi(w,z) \geq -\kappa(w,z)$$

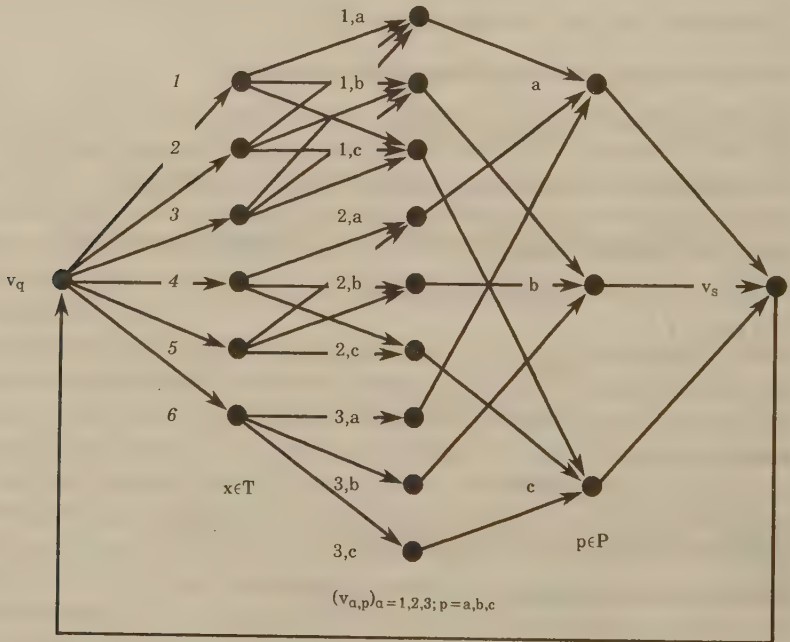
$$(c) \quad \forall (z,w) \in D: \quad \psi(w,z) \geq \lambda(w,z) \quad \text{enddef.}$$

Die Nebenbedingung (a) ist die Flußbedingung. Die Nebenbedingungen (b) und (c) fordern die Einhaltung der Grenzwerte für den Fluß.

Beispiel:

Es sind 6 Elementaroperationen $T := \{1, \dots, 6\}$, 3 Prozessoren $P := \{a, b, c\}$ und $t_{\max} = 3$ gegeben. Die Abbildung $t: T \rightarrow \mathbb{N}$ sei gegeben durch:

$x \in T$	1	2	3	4	5	6
$t(x)$	1	1	1	2	2	3



Im folgenden Satz wird gezeigt, daß UTAP mit FP gelöst werden kann.

4. Satz.

Gegeben seien die Bezeichnungen aus (1) und es sei $/P/ = s$. Es sei weiterhin die Abbildung $t : T \longrightarrow N$ eine zulässige Lösung von UTS. Die Kostenabbildungen von UTAP sei mit c_A bezeichnet.

Es sei V, D, λ, κ wie in (3). Definiere nun die Kostenabbildung c_F von FP:

$$c_F : D \longrightarrow \mathbf{R} \quad \text{mit } c_F(w, z) := \begin{cases} c_A(x, p) & \text{falls } (w, z) = (x, v_{t(x), p}) \\ 0 & \text{sonst} \end{cases}$$

Dann gibt es einen kostenminimalen Fluß ψ^* für FP bezüglich c_F und die Abbildung $a^* : E \longrightarrow \{0, 1\}$ mit

$$a^*(x, p) := \begin{cases} 1 & \text{falls } \psi^*(x, v_{t(x), p}) = 1 \\ 0 & \text{sonst} \end{cases}$$

eine Optimallösung von UTAP.

Beweis:

Vorerst wird eine kleine Hilfsaussage bewiesen:

$$(*) \quad E = T \times P$$

Begründung: $E = \{ (x, p) \in T \times P : t(x) < \min \{ t(y) : y \in N_R(x) \} \}$. Damit gilt " $\subset$ ".

Zu $E \supset T \times P$. Sei $(x, p) \in T \times P$ beliebig. Da t eine zulässige Lösung von UTS ist, gilt: $t(x) < \min \{ t(y) : y \in N_R(x) \}$. Hieraus folgt sofort $(x, p) \in E$.

Nun folgt der eigentliche Beweis: Er zerfällt in mehrere Teile:

(i) Es existiert eine zulässige Lösung von FP.

Mit Satz (1) existiert eine zulässige Lösung von UTAP. Aus dieser wird eine zulässige Lösung von FP konstruiert.

- (ii) Es gibt eine optimale Lösung von FP.

Da die Menge der zulässigen Lösungen von FP normbeschränkt ist, folgt aus der Existenz einer zulässigen Lösung auch die Existenz einer optimalen.

- (iii) Die Abbildung a^* ist zulässig für UTAP.

Nachprüfen der Nebenbedingungen von UTAP

- (iv) Die Abbildung a^* ist optimal für UTAP.

Falls a^* nicht optimal wäre, könnte eine zulässige Lösung von FP gefunden werden, deren Zielfunktionswert kleiner als der der Optimallösung ist.

Die Nebenbedingungen von UTS werden mit (Sa), (Sb), (Sc), die von UTAP mit (Aa), (Ab) und die von FP mit (Fa), (Fb), (Fc) bezeichnet.

Die Abbildung $t : T \longrightarrow \mathbb{N}$ ist eine zulässige Lösung von UTS. Nach Satz (1) existiert damit eine zulässige Lösung $a : E \longrightarrow \{0,1\}$ von UTAP. Mit dieser Abbildung a wird nun eine Abbildung $\psi : D \longrightarrow \mathbb{R}$ definiert:

$$\psi(w,z) := \begin{cases} /T/ & \text{falls } (w,z) = (v_s, v_q) \\ 1 & \text{falls } (w,z) \in \{v_q\} \times T \\ a(x,p) & \text{falls } (w,z) = (x, v_{t(x),p}) \text{ mit } x \in T \text{ und } p \in P \\ \sum_{x \in T_a} a(x,p) & \text{falls } (w,z) = (v_{a,p}, p) \text{ mit } a \in \{1, \dots, t_{\max}\} \\ \sum_{x \in T} a(x,p) & \text{falls } (w,z) \in P \times \{v_s\} \end{cases}$$

Nun wird die Zulässigkeit von ψ nachgeprüft.

Zuerst wird die Flußeigenschaft (Fa) betrachtet:

(Fa) für den Knoten v_q :

Nach Definition (3) gilt: $V_D(v_q) = \{v_s\}$ und $N_D(v_q) = T$. Damit ergibt sich:

$$-\sum_{z \in V_D(v_q)} \psi(z, v_q) + \sum_{z \in N_D(v_q)} \psi(v_q, z) = -\psi(v_s, v_q) + \sum_{x \in T} \psi(v_q, x) = -/T/ + \sum_{x \in T} 1 = 0$$

(Fa) für einen beliebigen Knoten $x \in T$:

Nach Definition (3) gilt: $V_D(x) = \{v_q\}$ und $N_D(x) = \{v_{t(x), p} : p \in P\}$. Damit ergibt sich:

$$-\sum_{z \in V_D(x)} \psi(z, x) + \sum_{z \in N_D(x)} \psi(x, z) = -\psi(v_q, x) + \sum_{p \in P} \psi(x, v_{t(x), p}) = -1 + \sum_{p \in P} a(x, p) =$$

(* a ist zulässig für (UTAP) *) $-1 + 1 = 0$

(Fa) für einen beliebigen Knoten $v_{\alpha, p}$ mit $\alpha \in \{1, \dots, t_{\max}\}$ und $p \in P$.

Nach Definition (3) gilt: $V_D(v_{\alpha, p}) = T_\alpha := t^{-1}(\alpha)$ und $N_D(v_{\alpha, p}) = \{p\}$. Damit ergibt sich:

$$-\sum_{z \in V_D(v_{\alpha, p})} \psi(z, v_{\alpha, p}) + \sum_{z \in N_D(v_{\alpha, p})} \psi(v_{\alpha, p}, z) = -\sum_{x \in T_\alpha} \psi(x, v_{\alpha, p}) + \psi(v_{\alpha, p}, p) =$$

$$-\sum_{x \in T_\alpha} a(x, p) + \psi(v_{\alpha, p}, p) = 0$$

(Fa) für einen beliebigen Knoten $p \in P$:

Nach Definition (3) gilt: $V_D(p) = \{v_{\alpha, p} : \alpha = 1, \dots, t_{\max}\}$ und $N_D(p) = \{v_s\}$. Damit ergibt sich:

$$-\sum_{z \in V_D(p)} \psi(z, p) + \sum_{z \in N_D(p)} \psi(p, z) = -\sum_{\alpha=1}^{t_{\max}} \psi(v_{\alpha, p}, p) + \psi(p, v_s) =$$

$$-\sum_{\alpha=1}^{t_{\max}} \sum_{x \in T_\alpha} a(x, p) + \psi(p, v_s) = (* (T_\alpha)_{\alpha=1, \dots, t_{\max}} \text{ ist eine Zerlegung von } T *)$$

$$-\sum_{x \in T} a(x, p) + \psi(p, v_s) = 0$$

(Fa) für den Knoten v_s :

Nach Definition (3) gilt: $V_D(v_s) = P$ und $N_D(v_s) = \{v_q\}$. Damit ergibt sich:

$$-\sum_{z \in V_D(v_s)} \psi(z, v_s) + \sum_{z \in N_D(v_s)} \psi(v_s, z) = -\sum_{p \in P} \psi(p, v_s) + \psi(v_s, v_q) =$$

$$-\sum_{p \in P} \sum_{x \in T} a(x, p) + /T/ = -\sum_{x \in T} \sum_{p \in P} a(x, p) + /T/ = (* \text{ a zulässig für (UTAP) } *)$$

$$-\sum_{x \in T} 1 + /T/ = 0$$

Insgesamt erfüllt ψ nun die Flußeigenschaft (Fa).

Zur Nebenbedingung (Fb) : $\forall (w, z) \in D : -\psi(w, z) \geq -\kappa(w, z)$.

Falls $(w, z) \in \{v_q\} \times T$ folgt: $-\psi(w, z) = -1 = -\kappa(w, z)$

Falls $(w, z) = (x, v_{t(x), p})$ mit $x \in T$ und $p \in P$ beliebig.

$$-\psi(w, z) = -a(x, p) \geq -\sum_{q \in P} a(x, q) = (* \text{ nach (Aa) } *) -1 = -\kappa(w, z)$$

Falls $(w, z) = (v_{\alpha, p}, p)$ mit $\alpha \in \{1, \dots, t_{\max}\}$ und $p \in P$ beliebig. Nach Definition von ψ gilt:

$$\psi(v_{\alpha, p}, p) = \sum_{x \in T_\alpha} a(x, p)$$

Ann.: $\psi(v_{\alpha, p}, p) > 1$.

Da a nur die Werte 0,1 annimmt gilt:

$$\exists x, y \in T_\alpha : x \neq y \text{ und } a(x, p) = 1 \text{ und } a(y, p) = 1$$

Weiterhin gilt: $x, y \in T_\alpha \Rightarrow t(x) = \alpha = t(y)$

Damit gilt insgesamt: $x, y \in T$ und $t(x) = t(y)$ und $x \neq y$ und $a(x, p) + a(y, p) = 2$.

Dies ist ein Widerspruch zu Nebenbedingung (Ab) von UTAP.

Somit gilt : $-\psi(v_{\alpha, p}, p) \geq -1 = -\kappa(v_{\alpha, p}, p)$

Falls $(w, z) \in (P \times \{v_s\}) \cup \{(v_s, v_q)\}$ ist (Fb) offensichtlich, da $\kappa(w, z) = \infty$

Damit erfüllt ψ die Nebenbedingung (b) von FP.

Zur Nebenbedingung (Fc): $\forall (w, z) \in D : \psi(w, z) \geq \lambda(w, z)$.

Falls $(w, z) \in \{v_q\} \times T$ gilt: $\psi(w, z) = 1 = \lambda(w, z)$

Da mit a auch ψ nicht negativ ist, gilt auch für die übrigen Pfeile (w, z) aus D:

$$\psi(w, z) \geq 0 = \lambda(w, z)$$

Damit ist $\psi : D \longrightarrow \mathbb{R}$ eine zulässige Lösung von FP.

Zu (ii): Es gibt eine Optimallösung von FP.

Es sei ψ eine beliebige zulässige Lösung von FP. Dann ist ψ in der 1-Norm nach oben und nach unten beschränkt, denn:

$$\|\psi\|_1 = \sum_{(w, z) \in D} \psi(w, z) \geq \min \{\lambda(w, z) : (w, z) \in D\} \geq 0$$

Definiere $D' := D \setminus ((P \times \{v_s\}) \cup \{(v_s, v_q)\})$

$$\|\psi\|_1 = \sum_{(w, z) \in D} \psi(w, z) \leq \sum_{(w, z) \in D'} \kappa(w, z) + \sum_{p \in P} \psi(p, v_s) + \psi(v_s, v_q) = (* (Fa) \text{ in } v_s *)$$

$$\sum_{(w, z) \in D'} 1 + 2 * \psi(v_s, v_q) = (* (Fa) \text{ in } v_q *) / D' / + 2 * \sum_{x \in T} \psi(v_q, x) \leq$$

$$/D' / + 2 * \sum_{x \in T} \kappa(v_q, x) = /D' / + 2 * /T /$$

Damit ist der zulässige Bereich von FP (interpretiert im $\mathbb{R}/D/$) beschränkt und abgeschlossen (wegen der " $=$ " und " $\leq$ " Restriktion). Somit folgt aus der Existenz einer zulässigen Lösung auch die Existenz einer Optimallösung für jede Kostenbewertung von FP (also auch für c_F).

Zu (iii): a^* ist zulässig für UTAP.

Dazu sind die Nebenbedingungen (Aa) und (Ab) von UTAP nachzuprüfen.

zu (Aa): (hier in äquivalenter Form) $\forall x \in T \exists! p \in P : a^*(x, p) = 1$

Sei nun $x \in T$ beliebig. Da ψ^* zulässig für FP ist, gilt:

$$1 = \lambda(v_q, x) \leq \psi^*(v_q, x) \leq \kappa(v_q, x) = 1$$

Optimallösungen von kapazitierten Flußproblemen sind nach (GARFINKEL-72/ p 70ff) ganzzahlig. Mit der Nichtnegativität von ψ^* folgt aus der Flußbedingung, daß es genau ein $z \in N_D(x) : \psi^*(x, z) = 1$ gibt. Da $N_D(x) = \{v_{t(x), p} : p \in P\}$, gibt es genau ein $p \in P : \psi^*(x, v_{t(x), p}) = 1$. Aus der Definition von a^* folgt die Behauptung.

zu (Ab): $\forall p \in P \forall x, y \in T : (t(x) = t(y) \text{ und } x \neq y) \Rightarrow a^*(x, p) + a^*(y, p) \leq 1$

Sei nun $x, y \in T$ beliebig mit $t(x) = t(y)$ und $x \neq y$

Ann.: $\exists p \in P : a^*(x, p) + a^*(y, p) > 1$. Nach Definition von a^* ist diese Annahme äquivalent zu: $\exists p \in P : a^*(x, p) = 1$ und $a^*(y, p) = 1$.

Nach Definition von a^* folgt dann: $\psi^*(x, v_{t(x), p}) = 1$ und $\psi^*(y, v_{t(x), p}) = 1$. Da ψ^* die Flußeigenschaft erfüllt, muß die Summe der Flußwerte auf den Nachfolgern von $v_{t(x), p}$ mindestens 2 sein. Da die Nachfolgermenge nur aus dem Knoten p besteht, muß der Flußwert $\psi^*(v_{t(x), p}, p)$ mindestens 2 sein. Dies ist jedoch ein Widerspruch zur Flußgrenzenfestlegung (Fb) für $(v_{t(x), p}, p)$ da $\kappa(v_{t(x), p}, p) = 1$ ist.

Damit gilt (Ab).

Insgesamt ist a^* eine zulässige Lösung von UTAP.

Als letztes muß noch die Optimalität von a^* gezeigt werden. Dazu wird angenommen, daß es eine zulässige Lösung $a' : E \longrightarrow N_0$ von UTAP gibt, für die gilt:

$$c_A T a' < c_A T a^*$$

Aus der Abbildung a' kann wie vorher aus a ein zulässiger Fluß $\psi' : D \longrightarrow \mathbb{R}$ konstruiert werden.

Dann gilt:

$$c_F^T \Psi' = \sum_{(w,z) \in D} c_F(w,z) * \Psi'(w,z) = (* c_F \text{ ist nur auf } \{x\} \times N_D(x) : x \in T \text{ ungleich Null} *)$$

$$\sum_{x \in T} \sum_{z \in N_D(x)} c_F(w,z) * \Psi'(w,z) = \sum_{x \in T} \sum_{p \in P} c_F(x, v_{t(x),p}) * \Psi'(x, v_{t(x),p}) =$$

$$\sum_{x \in T} \sum_{p \in P} c_A(x,p) * a'(x,p) = (* \text{ mit } E = T \times P *) \quad c_A^T a' < c_A^T a^* =$$

$$\sum_{x \in T} \sum_{p \in P} c_A(x,p) * a^*(x,p) = \sum_{x \in T} \sum_{p \in P} c_F(x, v_{t(x),p}) * \Psi^*(x, v_{t(x),p}) =$$

$$\sum_{x \in T} \sum_{z \in N_D(x)} c_F(w,z) * \Psi^*(w,z) = \sum_{(w,z) \in D} c_F(w,z) * \Psi^*(w,z) = c_F^T \Psi^*$$

Dies ist ein Widerspruch zur Optimalität von Ψ^*

qed

Der vorangegangene Satz (4) eröffnet eine unmittelbare Möglichkeit UTAP mit bekannten Netzflußalgorithmen zu lösen. Die Formulierung von FP mag auf den ersten Blick zu umfangreich sein, denn die Knoten P könnten auch weggelassen werden. Falls verschiedenartige Prozessoren und verschiedene Elementaroperationen mit $\tau = 1$ betrachtet werden, entsteht eine Verallgemeinerung von UTAP. Die Menge E der möglichen Zuordnungen ist dann in der Regel eine echte Teilmenge von $T \times P$. Der Netzflußansatz (3) ist auch fähig, dieses verallgemeinerte Problem zu lösen, da für die totale Unimodularität bereits $\tau = 1$ und $R_d = \emptyset$ ausreichend ist.

9. Zuordnungsgraphen

In diesem Kapitel wird die Lösung von GAP vorbereitet. Vorerst wird über den möglichen Zuordnungen E von Elementaroperationen T an Prozessoren P der Zuordnungsgraph definiert. Danach wird ein Existenzsatz für LAP bewiesen, der besagt, daß eine Optimallösung von LAP genau dann existiert, wenn die Stabilitätszahl des Zuordnungsgraphen gleich der Anzahl der Elementaroperationen ist. Daran anschließend wird ein neues Problem CLP definiert, mit dessen Hilfe LAP_{con} gelöst werden kann. Nun wird das Problem CLP mit Schlupfvariablen auf Standardform CLP_{stan} gebracht. Für CLP_{stan} wird folgender wichtiger Satz bewiesen: Die zulässigen Basislösungen von CLP_{stan} sind immer dann ganzzahlig, wenn der Zuordnungsgraph perfekt /GOLUMBIC-80/ ist. Der Beweis dieses Satzes benützt das Perfektheitskriterium von /CHVATAL-75/ in der Form von /GOLUMBIC-80/ p 63. Die Perfektheit des Zuordnungsgraphen ist jedoch nicht notwendig für die Ganzzahligkeit der Basislösungen von CLP_{stan} . Dies zeigt ein Gegenbeispiel.

Vorerst sei an die für dieses Kapitel relevanten Bezeichnungen erinnert:

1. Bezeichnungen.

T	Menge der Elementaroperationen
$R \subset v(T)$	Zeitliches Nacheinander der Elementaroperationen
$R_d \subset R$	Datentransfers zwischen Elementaroperationen
$\tau : T \longrightarrow N$	Länge des Ausführungsintervalls
t_{max}	Maximaler Ausführungsbeginn
$t : T \longrightarrow N$	Anfangszeitpunkte
$\Phi = \{\phi_1, \dots, \phi_k\}$	Menge der Ressourcen
$\mu : T \longrightarrow N_0^k$	Ressourcenanforderungen
$s \in N_0^k$	Gesamtverfügbarkeiten
P	Menge der Prozessoren
$W \subset v(P)$	Verschaltung der Prozessoren
W_l	Leitungen mit limitiertem Verzögerungswert
W_f	Leitungen mit festem Verzögerungswert
$\eta : P \longrightarrow N_0^k$	Bereitstellung von Ressourcen
$\delta : P \longrightarrow]0, 1]$	Geschwindigkeiten
$r_p : R_d \longrightarrow Z$	Verzögerungswert des Datentransfers
$\xi : W \longrightarrow N_0$	Verzögerungswert (schränke)

$E \subset T \times P$	Mögliche Zuordnungen
$I_p : p \in P$	Inkompatibilitäten

Für E und $I_p : p \in P$ gilt nach Definition (7.4)

$$(x, p) \in E \Leftrightarrow \begin{cases} \text{(i)} & \mu(x) \leq \eta(p) \leq s \\ \text{(ii)} & t(x) + \tau(x) * \delta(p) \leq \min \{ t(y) : y \in N_R(x) \} \end{cases}$$

$$(x, y) \in I_p \Leftrightarrow \begin{cases} t(y) < t(x) + \tau(x) * \delta(p) & \text{falls } t(x) \leq t(y) \text{ und } x \neq y \\ t(x) < t(y) + \tau(y) * \delta(p) & \text{falls } t(x) > t(y) \end{cases}$$

Über der Menge E wird nun wiederum eine Graphenstruktur definiert. Die Kanten über der Knotenmenge E werden mit $[(x, p), (y, q)]$ bezeichnet.

2. Definition.

Über der Menge der möglichen Zuordnungen E wird nun der ungerichtete *Zuordnungsgraph* (E, Z) definiert mit:

$$Z := Z = \cup Z_{\leq} \text{ wobei}$$

$$Z = \bigcup_{x \in T} \{[(x, p), (x, q)] : (x, p), (x, q) \in E \text{ und } p \neq q\}$$

$$Z_{\leq} := \bigcup_{p \in P} \{[(x, p), (y, p)] : (x, p), (y, p) \in E \text{ und } (x, y) \in I_p\} \cup$$

$$\text{sym} \{[(x, p), (y, q)] : (x, p), (y, p) \in E \wedge (x, y) \in R_d \wedge (p, q) \in v(P) \setminus W\} \cup$$

$$\text{sym} \{[(x, p), (y, q)] : (x, p), (y, p) \in E \wedge (x, y) \in R_d \wedge (p, q) \in W_f \wedge \xi(p, q) \neq r_p(x, y)\} \cup$$

$$\text{sym} \{[(x, p), (y, q)] : (x, p), (y, p) \in E \wedge (x, y) \in R_d \wedge (p, q) \in W_l \wedge \xi(p, q) < r_p(x, y)\}$$

enddef.

Da die Graphen $I_p : p \in P$ symmetrisch sind, ist auch Z symmetrisch. Damit ist (E, Z) ein ungerichteter Graph. Eine Kante besteht genau dann zwischen zwei Elementen (x, p) und (y, q) aus E , wenn für eine zulässige Lösung $a : E \longrightarrow \{0, 1\}$ nicht $a(x, p) > 0$ und $a(y, q) > 0$ gelten darf. Damit besteht ein enger Zusammenhang zwischen stabilen Mengen in (E, Z) und zulässigen Lösungen von LAP. Dies zeigt der folgende

3. Satz.

Gegeben seien die Bezeichnungen aus (1) sowie LAP aus (7.5) und der Zuordnungsgraph (E, Z) aus (3). Dann existiert eine Optimallösung von LAP genau dann, wenn gilt:

$$\sigma(Z) = /T/$$

D.h. die Stabilitätszahl von (E, Z) ist gleich der Anzahl der Elementaroperationen.

Beweis:

Die Nebenbedingungen von LAP werden mit $(La), (Lb), (Lc)$ bezeichnet.

"nur dann": Es sei $a : E \rightarrow \{0, 1\}$ eine zulässige Lösung von LAP.

Behauptung: $S := \{(x, p) \in E : a(x, p) = 1\}$ ist eine $/T/$ -elementige stabile Menge in (E, Z) ; d.h. die von S induzierte Relation $Z(S)$ ist leer.

Begründung: Es seien (x, p) und (y, q) beliebig in S und $(x, p) \neq (y, q)$

Ann.: $[(x, p), (y, q)] \in Z = \Rightarrow x = y$ und $a(x, p) \geq 1$ und $a(x, q) \geq 1$

Betrachte nun Nebenbedingung (La) für den Knoten $x \in T$.

$$\sum_{n \in N_E(x)} a(x, n) \geq (* \text{ Nichtnegativität von } a *) a(x, p) + a(x, q) \geq 2$$

Dies ist ein Widerspruch zu Nebenbedingung (La) .

Ann.:

$$[(x, p), (y, q)] \in \bigcup_{p \in P} \{[(x, p), (y, p)] : (x, p), (y, p) \in E \text{ und } (x, y) \in I_p\}$$

Es folgt: $p = q$ und $(x, y) \in I_p$ und $a(x, p) \geq 1$ und $a(y, p) \geq 1$. Dies liefert jedoch einen Widerspruch zu (Lb) , da $(x, y) \in I_p$ und $a(x, p) + a(y, p) \geq 2$

Ann.:

$$[(x, p), (y, q)] \in \text{sym} \{[(w, n), (z, p)] : (w, n), (z, p) \in E \text{ und } (w, z) \in R_d \text{ und } (n, p) \in v(P) \setminus W\}$$

Es folgt: $((x,y) \in R_d \wedge (p,q) \in v(P) \setminus W) \vee ((y,x) \in R_d \wedge (q,p) \in v(P) \setminus W)) \wedge a(x,p) \geq 1 \wedge a(y,q) \geq 1$

Damit ergibt sich jedoch ein Widerspruch zu (Lc)(1), da

$$(((x,y) \in R_d \wedge (p,q) \in v(P) \setminus W) \vee ((y,x) \in R_d \wedge (q,p) \in v(P) \setminus W)) \wedge a(x,p) + a(y,q) \geq 2$$

Ann.:

$$[(x,p),(y,q)] \in \text{sym} \{[(w,n),(z,p)] : (w,n), (z,p) \in E \wedge (w,z) \in R_d \wedge (n,p) \in W_f \wedge \xi(n,p) \neq r_p(w,z)\} \Rightarrow$$

$$(((x,y) \in R_d \wedge (p,q) \in W_f \wedge \xi(p,q) \neq r_p(x,y)) \vee ((y,x) \in R_d \wedge (q,p) \in W_f \wedge \xi(q,p) \neq r_p(y,x))) \wedge a(x,p) \geq 1 \wedge a(y,q) \geq 1 \Rightarrow$$

daus ergibt sich ein Widerspruch zu (Lc)(2), da

$$(((x,y) \in R_d \wedge (p,q) \in W_f \wedge \xi(p,q) \neq r_p(x,y)) \vee ((y,x) \in R_d \wedge (q,p) \in W_f \wedge \xi(q,p) \neq r_p(y,x))) \wedge a(x,p) + a(y,q) \geq 2$$

Ann.:

$$[(x,p),(y,q)] \in \text{sym} \{[(w,n),(z,p)] : (w,n), (z,p) \in E \wedge (w,z) \in R_d \wedge (n,p) \in W_1 \wedge \xi(n,p) < r_p(w,z)\} \Rightarrow$$

$$(((x,y) \in R_d \wedge (p,q) \in W_1 \wedge \xi(p,q) < r_p(x,y)) \vee ((y,x) \in R_d \wedge (q,p) \in W_1 \wedge \xi(q,p) < r_p(y,x))) \wedge a(x,p) \geq 1 \wedge a(y,q) \geq 1 \Rightarrow$$

daus ergibt sich ein Widerspruch zu (Lc)(3), da

$$(((x,y) \in R_d \wedge (p,q) \in W_1 \wedge \xi(p,q) < r_p(x,y)) \vee ((y,x) \in R_d \wedge (q,p) \in W_1 \wedge \xi(q,p) < r_p(y,x))) \wedge a(x,p) + a(y,q) \geq 2$$

Damit ist S stabil. Es gilt:

$$/S/ = / \{ (x,p) \in E : a(x,p) > 0 \} / = / \{ (x,p) \in E : a(x,p) = 1 \} / =$$

$$\sum_{(x,p) \in E} a(x,p) = \sum_{x \in T} \sum_{p \in N_E(x)} a(x,p) = \sum_{x \in T} 1 = /T/$$

Damit ist S eine /T/ elementige Menge in (E,Z). $\Rightarrow \sigma(Z) \geq /T/$

Ann.: $\sigma(Z) > /T/$. Wähle eine stabile Menge S mit $/S/ > /T/$. Es gilt $S \subset E \subset T \times P$

Da $/S/ > /T/$, gibt es ein $x \in T$ und $p, q \in P$ mit $p \neq q : (x,p), (x,q) \in S$

Nach Definition von Z_* ist $(\{x\} \times N_E(x))$ vollständig in Z . Damit enthält S zwei Elemente einer vollständigen Menge. Dies ist ein Widerspruch zur Stabilität von S .

Damit gilt insgesamt: $\sigma(Z) = /T/$

"immer dann": Sei S eine stabile Menge in (E, Z) mit $/S/ = /T/$. Definiere eine Abbildung $a: E \longrightarrow N_0$:

$$a(x, p) := \begin{cases} 1 & \text{falls } (x, p) \in S \\ 0 & \text{sonst} \end{cases}$$

Behauptung: Diese Abbildung ist für LAP zulässig.

zu (La) (hier in äquivalenter Form) $\forall x \in T \exists_1 p \in P: a(x, p) = 1$

Es sei $x \in T$ beliebig. Da $(\{x\} \times N_E(x))$ eine Clique in Z ist, gilt:

$$(+)\quad /S \cap (\{x\} \times N_E(x)) / \leq 1$$

Damit kann eine Abbildung $f: S \longrightarrow T$ mit $f(y, p) := y$ definiert werden. Nach

(+) ist f injektiv. Mit $/S/ = /T/$ folgt, daß f sogar bijektiv ist. Hieraus folgt:

$$\forall x \in T \exists_1 p \in P: (x, p) \in S \Rightarrow \forall x \in T \exists_1 p \in P: a(x, p) = 1$$

Damit ist (La) gezeigt.

(Lb) und (Lc) ergeben sich unmittelbar aus der Stabilität von S und der Definition von Z .

Damit ist die Abbildung $a: E \longrightarrow \{0, 1\}$ eine zulässige Lösung für LAP.

Alle zulässigen Lösungen von LAP liegen in $\{0, 1\}^{E/}$. Somit folgt aus der Existenz einer zulässigen Lösung auch die Existenz einer optimalen. qed.

Dieser Satz bildet die Grundlage für die algorithmische Lösung von GAP. Es werden nacheinander alle stabilen Mengen mit $/T/$ Knoten ermittelt. Daraus werden Lösungen von LAP abgeleitet. Für diese Lösungen werden die Gesamtressourcenbeschränkungen nachgeprüft und der Zielfunktionswert ermittelt. Ein effizientes Verfahren zur Bestimmung aller stabiler Mengen findet man beispielsweise in /LOUKAKIS-81/.

Da LAP ein lineares ganzzahliges Programm ist, liegt die Frage nahe, ob und wann eine Optimallösung von LAP_{con} stets ganzzahlig ist. In diesem Fall kann eine optimale Lösung von LAP mit dem Simplexalgorithmus ermittelt werden. Zur Klärung der obigen Fragestellung wird nun ein kontinuierliches lineares Programm definiert, dessen zulässige Lösungen ebensolche für LAP_{con} induzieren.

4. Cliquenproblem. CLP - "clique problem"

Gegeben seien die Bezeichnungen aus (1) sowie die Menge $\underline{C}$ aller maximaler Cliques von (E, Z) . Es sei die Abbildung $c: E \longrightarrow \mathbb{R}$ eine beliebige Kostenbewertung von E . Gesucht ist eine Abbildung $a: E \longrightarrow \mathbb{R}$, sodaß

$$c^T a \longrightarrow \min$$

unter den Nebenbedingungen:

$$(a) \quad \forall x \in T: \quad \sum_{p \in N_E(x)} a(x, p) = 1$$

$$(b) \quad \forall C \in \underline{C}: \quad \sum_{(x, p) \in C} a(x, p) \leq 1$$

$$(c) \quad \forall (x, p) \in E: \quad a(x, p) \geq 0$$

enddef

Der folgende Satz zeigt, daß die zulässigen Lösungen von CLP auch zulässig für LAP_{con} sind.

5. Satz

Gegeben sei LAP_{con} aus (7.6) und CLP aus (4). Es sei $a: E \longrightarrow \mathbb{R}$ eine zulässige Lösung von CLP. Dann ist diese Abbildung a auch zulässig für LAP_{con} .

Beweis:

Die Nebenbedingungen von LAP_{con} bzw CLP werden mit (La),(Lb),(Lc),(Le) bzw (Ca),(Cb),(Cc) bezeichnet.

(La) und (Ca) sowie (Cc) und (Le) sind identisch.

Nun erfülle die Abbildung $a: E \rightarrow \mathbb{R}$ die Bedingung (Cb). z.z. (Lb), (Lc)

$$\text{zu (Lb): } \forall p \in P \quad \forall (x,y) \in I_p: \quad a(x,p) + a(y,p) \leq 1$$

Sei $p \in P$ und $(x,y) \in I_p$ so gilt: $[(x,p),(y,p)] \in Z$. Damit gibt es eine Clique $C \in \underline{C}$ mit $(x,p),(y,p) \in C$. Damit gilt insgesamt mit (Cc):

$$a(x,p) + a(y,p) \leq \sum_{(z,q) \in C} a(z,q) \leq 1$$

$$\text{zu (Lc)(1): } \forall (x,y) \in R_d \quad \forall (p,q) \in v(P) \setminus W: \quad (x,p), (y,q) \in E \Rightarrow a(x,p) + a(y,q) \leq 1$$

Sei $(x,y) \in R_d$, $(p,q) \in v(P) \setminus W$ mit $(x,p), (y,q) \in E$. So gilt nach Definition von $Z_{\leq}$, daß $[(x,p),(y,q)] \in Z_{\leq} \subset Z$. Damit gibt es eine Clique $C \in \underline{C}$ mit $(x,p),(y,q) \in C$. Insgesamt gilt mit (Cc):

$$a(x,p) + a(y,q) \leq \sum_{(z,n) \in C} a(z,n) \leq 1$$

Die Argumentation für (Lc)(2),(3) ist analog.

Nun sind alle Nebenbedingungen von LAP_{con} gezeigt.

qed

Nun folgt der wichtige Begriff der Perfektheit von Graphen. Eine weitreichende Übersicht über diese Graphenklasse bietet /GOLUMBIC-80/. Das nächste Ziel ist der Beweis folgender Tatsache: Das Problem CLP wird durch Schlupfvariable auf Gleichheitsrestriktionen gebracht. Falls der Zuordnungsgraph perfekt ist, ist jede Basislösung des obigen Problems ganzzahlig.

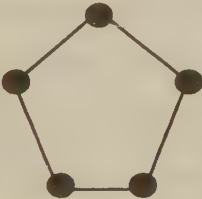
Es gibt mehrere äquivalente Definitionen für perfekte Graphen /GOLUMBIC-80/. Für die hier verfolgten Zwecke wählen wir folgende

6. Definition.

Ein ungerichteter Graph (X, R) heißt genau dann *perfekt*, wenn gilt:

$$\forall Y \subset X : |Y| \leq \sigma(R(Y)) * \omega(R(Y)) \quad \text{enddef.}$$

In einem perfekten Graphen gilt für jede Teilmenge $Y \subset X$, daß das Produkt der Stabilitätszahl und der Cliquenzahl des von Y induzierten Untergraphen nie kleiner ist als die Knotenzahl von Y . Beispielweise ist ein ungerader Zyklus von mehr als 3 Knoten nicht perfekt:



Der Zyklus mit 5 Knoten hat Stabilitätszahl und Cliquenzahl 2 aber 5 Knoten.

7. Satz.

Gegeben sei mit den entsprechenden Bezeichnungen CLP aus (4). Es gelte weiterhin:

$$(+)\quad \forall x \in T \exists p \in P : (x, p) \in E$$

Dann gilt: Falls der Zuordnungsgraph (E, Z) aus (2) perfekt ist, so hat das auf Gleichheitsrestriktionen erweiterte CLP nur ganzzahlige zuläss. Basislösungen.

Beweis:

Skizze:

- CLP wird auf Standardform gebracht.
- Jede zulässige Basislösung des Standardproblems kann auf einen Extrempunkt des reinen Ungleichungsproblem projiziert werden.

- Mit dem Satz von /CHVATAL-75/ folgt, daß der Extrempunkt ganzzahlig ist.
- Auch die zulässige Basislösung (als Rückprojektion) ist ganzzahlig.

Definiere für alle $x \in T$: $C_x := \{(x,p) \in T \times P : p \in N_E(x)\}$

zu (i): Es wird eine weitere Abbildung $b : \underline{C} \longrightarrow \mathbb{R}$ als Schlupfabildung zu CLP hinzugenommen. Damit kann CLP_{stan} definiert werden:

Gegeben sind die Menge der maximalen Cliques $\underline{C}$ und $c : E \longrightarrow \mathbb{N}_0$.

Gesucht sind Abbildungen $a : E \longrightarrow \mathbb{R}$ und $b : \underline{C} \longrightarrow \mathbb{R}$, sodaß

$cTa \longrightarrow \min$

unter den Nebenbedingungen:

$$(a) \quad \forall x \in T: \sum_{(x,p) \in C_x} a(x,p) = 1$$

$$(b) \quad \forall C \in \underline{C}: \sum_{(x,p) \in C} a(x,p) + b(C) = 1$$

$$(c) \quad \forall (x,p) \in E: a(x,p) \geq 0$$

$$(d) \quad \forall C \in \underline{C}: b(C) \geq 0$$

Die Abbildungen a und b werden mit den Vektoren $a \in \mathbb{R}^E$ bzw $b \in \mathbb{R}^{\underline{C}}$ identifiziert.

Nun wird gezeigt, daß die Matrix der Nebenbedingungen von CLP_{stan} Höchstrang hat. Wähle nach (+) eine Teilmenge $E' \subset E$ mit

- $\forall (x,p), (y,q) \in E': x \neq y$
- $\forall x \in T \exists p \in P : (x,p) \in E'$

Die Teilmenge E' hat in der 1. Komponente paarweise verschiedene Elemente und es gilt: $|E'| = |T|$. Damit ergibt sich (nach event. Umordnung der Spalten) folgende Gestalt für die Matrix M der Nebenbedingungen von CLP_{stan} :

	$E \setminus E'$	E'	$\underline{C}$
T	*	1, ... 1	0
$\underline{C}$	*	*	1 1 1 0 1 1

Damit hat der hintere Teil der Matrix untere Dreiecksgestalt und Diagonalelemente mit Wert 1. Die Matrix M hat folgende schematisierte Form:

$$M = \begin{array}{|c|c|} \hline M_1 & 0 \\ \hline M_2 & I_{\underline{C}} \\ \hline \end{array}$$

Definiere nun $n := |E| + |\underline{C}|$.

Betrachte nun das durch M in $\mathbb{R}^n$ festgelegte Polytop P_1 :

$$P_1 := \{x \in \mathbb{R}^n : Mx = 1 \text{ und } x \geq 0\}$$

Es sei nun $(a, b) \in \mathbb{R}^n$ eine zulässige Basislösung von CLP_{stan} . Nach /LUENBERGER-72/ p 21 ist (a, b) ein Extrempunkt des Polytops P_1 .

Definiere nun $m := |E|$. Im $\mathbb{R}^m$ wird nun ein weiteres Polytop P_2 definiert:

$$P_2 := \{x \in \mathbb{R}^m : M_2 x \leq 1, x \geq 0\} \quad \text{Es gilt } a \in P_2, \text{ da } M_2 a = 1 - b \leq 1.$$

Nun wird gezeigt, daß a ein Extrempunkt von P_2 ist:

Ann.: a ist kein Extrempunkt von P_2 . Dann gibt es ein

$$0 < \lambda < 1 \text{ und } x \neq y \in P_2 :$$

$$(+)\quad a = \lambda * x + (1 - \lambda) * y.$$

Definiere nun $b_x := 1 - M_2 x$ und $b_y := 1 - M_2 y$. Zeige nun, daß gilt:

$$(\#)\quad (x, b_x) \in P_1 \text{ und } (y, b_y) \in P_1.$$

Ann.: $\exists z \in T: m_{1z} T x < 1$. Hierbei bezeichnet m_{1z} die z-te Zeile von M_1 .

$$\text{Es gilt mit } (a, b) \in P_1: 1 = m_{1z} T a = m_{1z} T (\lambda * x + (1 - \lambda) * y) =$$

$$\lambda * m_{1z} T x + (1 - \lambda) * m_{1z} T y < \lambda * 1 + (1 - \lambda) * m_{1z} T y \leq \lambda * 1 + (1 - \lambda) * 1 = 1.$$

Dies ist ein Widerspruch.

$$\text{Somit gilt: } m_{1z} T x = 1 \text{ und somit } M_1 x = 1.$$

$$\text{Analog zeigt man: } M_1 y = 1$$

Mit der Definition von b_x und b_y folgt sofort die Behauptung (#)

Betrachte nun:

$$\lambda * b_x + (1 - \lambda) * b_y = \lambda * (1 - M_2 x) + (1 - \lambda) * (1 - M_2 y) =$$

$$\lambda * 1 - \lambda * M_2 x + 1 - M_2 y - \lambda * 1 + \lambda * M_2 y = 1 - M_2 (\lambda * x + (1 - \lambda) * y) =$$

$$1 - M_2 a = b.$$

Damit kann (mit (+)) (a, b) als Konvexkombination der Elemente (x, b_x) und (y, b_y) aus P_1 dargestellt werden. - Widerspruch zur Extremaleigenschaft -

Damit ist gezeigt, daß a ein Extrempunkt von P_2 ist.

Benutze nun den Satz von /CHVATAL-75/ in der Form von /GOLUMBIC-80/ p 63:

$$P_{12} := \text{konvexe Hülle } \{x \in P_2 : x \text{ ganzzahlig}\}$$

$\underline{C}$ ist die Menge aller maximaler Cliques in (E, Z) . Die Matrix M_2 heißt (in der Terminologie von Golumbic) Cliquesmatrix von (E, Z) .

Der Satz von /CHVATAL-75/ besagt:

$$(E, Z) \text{ perfekt} \Leftrightarrow P_{12} = P_2$$

Da (E, Z) als perfekt vorausgesetzt wurde, ist der Extrempunkt a ganzzahlig. Da M_2 eine ganzzahlige Matrix ist, ist auch $b = 1 - M_2 a$ ganzzahlig. Damit ist die Ganzzahligkeit der zulässigen Basislösung (a, b) gezeigt. qed.

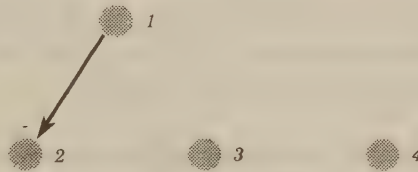
Nun wird gezeigt, daß die Perfektheit von (E, Z) nicht notwendig für die Ganz-zahligkeit der Basislösungen von CLP_{stan} ist.

8. Beispiel.

Gegeben seien 4 Elementaroperationen $T := \{1, 2, 3, 4\}$ zusammen mit einer Resource ϕ_1 , die in 5 Einheiten vorhanden ist; d.h. $s := 5$. Die Elementaroperationen haben folgende Eigenschaften:

$x \in T$	1	2	3	4
$t(x)$	1	3	2	3
$\tau(x)$	2	2	2	2
$\mu(x)$	2	1	3	2

Die azyklische Relation R ist identisch mit den Datentransfers R_d :



Die Ausführungsintervalle der Elementaroperationen zeigt das nächste Bild:

Elementarop \ Zeitintervall	[1,2[	[2,3[	[3,4[	[4,5[
Operation 1	x	x		
Operation 2			x	x
Operation 3		x	x	
Operation 4		x	x	

Die Operation 2 beginnt erst dann, wenn Operation 1 beendet ist.

Für die pro Zeitintervall benutzten Ressourcen ergibt sich:

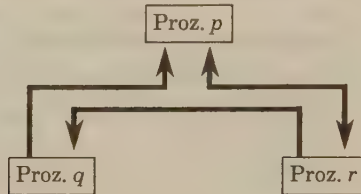
Benutzte Res. \ Zeitintervall	[1,2[	[2,3[	[3,4[	[4,5[
1. Einheit von ϕ_1	x	x	x	x
2. Einheit von ϕ_1	x	x	x	x
3. Einheit von ϕ_1		x	x	x
4. Einheit von ϕ_1		x	x	
5. Einheit von ϕ_1			x	

Damit werden in keinem Zeitintervall mehr als 5 Einheiten der Resource ϕ_1 benötigt. Insgesamt ist t eine für (RCS) zulässige Lösung.

Gegeben seien weiterhin 3 Prozessoren $P := \{p, q, r\}$ mit folgenden Eigenschaften:

$\pi \in P$	p	q	r
$\delta(x)$	1	1	1
$\eta(x)$	3	2	1

Weiterhin sei folgende Verdrahtung der Prozessoren gegeben:



Für die Pfeilmenge W ergibt sich: $W = \{(p,r), (r,p), (r,q), (q,p)\}$

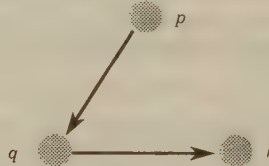
Weiters wird vorausgesetzt: $W_f = \emptyset$ und $W_l = W$. Der Verzögerungswert wird für alle Leitungen aus W identisch 1 gesetzt, d.h.

$\xi: W \rightarrow \mathbb{Z}, \xi \equiv 1$

Da $\delta \equiv 1$, sind die Verzögerungswerte r_π für alle $\pi \in \{p, q, r\}$ identisch:

$r_\pi(1,2) = t(2) - Lt(1) + \tau(1) * \delta(1) = 3 - (1 + 2 * 1) = 0$

Damit ergibt sich der Graph $v(P) \setminus W = \{(p,q), (q,r)\}$:



Mit diesen Definitionen kann die Menge E der möglichen Zuordnungen bestimmt werden:

$E = \{(1,p), (1,q), (2,p), (2,q), (2,r), (3,p), (3,q), (4,p), (4,q)\}$

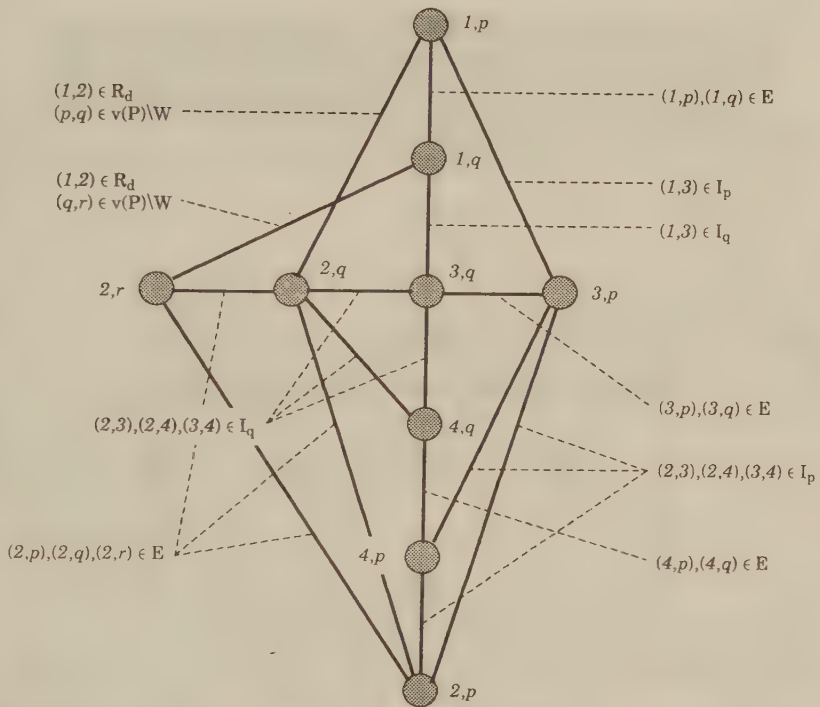
Die zeitlichen Überlappungen (Inkompatibilitäten) sind für alle Prozessoren identisch:

$\forall \pi \in P: I_\pi = \text{sym} \{ (1,3), (2,3), (3,4), (2,4) \}$

Da $W_f = \emptyset$ entfällt Nebenbedingung (c)(2) von LAP_{con} .

Da $r_\pi \equiv 0$ und $\xi \equiv 1$ entfällt auch Nebenbedingung (c)(3) von LAP_{con} .

Damit ergibt sich für den Zuordnungsgraphen (E, Z) :



Für die Matrix M von CLP_{stan} ergibt sich:

[illegible]

Hierbei bezeichnet der Vektor a folgenden Variable:

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_7	a_9
$1,p$	$1,q$	$2,p$	$2,q$	$2,r$	$3,p$	$3,q$	$4,p$	$4,q$

Aus den ersten vier Gleichungen ergibt sich:

$$a_2 = 1 - a_1$$

$$a_5 = 1 - a_3 - a_4$$

$$a_7 = 1 - a_6$$

$$a_9 = 1 - a_8$$

$$b_7 = b_8 = b_9 = b_{10} = 0$$

Damit ergibt sich für das Gleichungssystem

a_1	a_3	a_4	a_6	a_8	b_1	b_2	b_3	b_4	b_5	b_6	
1			1		1						1
	1			1	1						1
1		1					1			a	= 1
-1	-1	-1						1		b	-1
			1	-1	-1				1		-1
-1			-1							1	-1

$$a_1 + a_6 + b_1 = 1 \text{ und } a_1 + a_6 - b_6 = 1 \Rightarrow (* b_1, b_6 \geq 0 *)$$

$$a_1 + a_6 = 1 \text{ und } b_1 = b_6 = 0$$

$$a_6 = 1 - a_1$$

Damit ergibt sich für das Gleichungssystem

a_1	a_3	a_4	a_8	b_2	b_3	b_4	b_5	
-1	1		1	1				0
1		1			1			a = 1
-1	-1	-1				1		b -1
1		1	-1				1	0

Zähle nun die vier Zeilen der Matrix zusammen:

$$a_4 + b_2 + b_3 + b_4 + b_5 = 0$$

Da alle Größen als nichtnegativ vorausgesetzt sind, folgt:

$$a_4 = b_2 = b_3 = b_4 = b_5 = 0$$

Hiemit folgt:

$$a_1 = 1$$

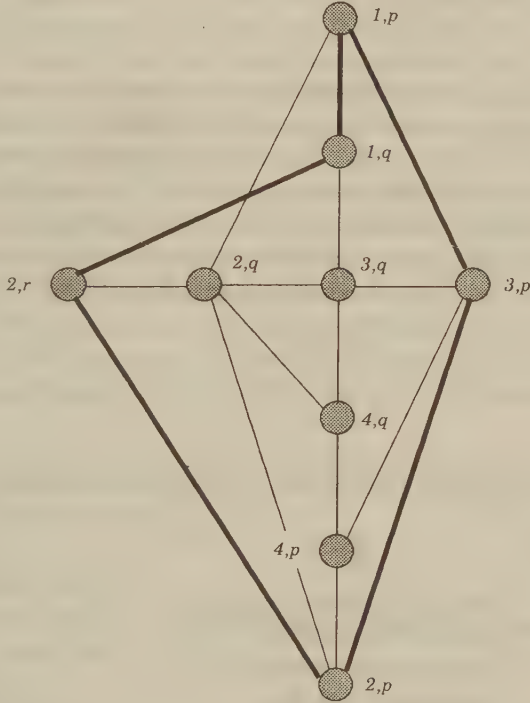
$$a_8 = 1$$

$$a_3 = 0$$

Somit ist das Gleichungssystem CLP_{stan} eindeutig lösbar. CLP_{stan} hat damit genau eine Basislösung:

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}
1	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0

Damit sind sämtliche Basislösungen von CLP_{stan} ganzzahlig. Jedoch der Zuordnungsgraph enthält einen ungeraden Zyklus als induzierten Untergraphen:



10. Lösung von GAP und Verzögerungsoptimierung

Falls die Gesamtzahl der Ressourcen, die die Prozessoren bereitstellen, beschränkt ist, wird LAP aus (7.5) zu GAP aus (7.4). Durch die Nichtlinearität von GAP kann man nicht erwarten, daß simplexähnliche Algorithmen als Lösungsverfahren in Frage kommen. Der Satz (9.3) bildet die Grundlage zur Lösung von GAP. Es werden im Zuordnungsgraph alle stabilen Mengen mit $/T/$ Knoten untersucht. Aus denjenigen Mengen, die zusätzlich die Gesamtressourcenrestriktion erfüllen, wird die mit dem geringsten Zielfunktionswert ausgewählt. Algorithmische Grundlage für die Lösung von GAP ist das derzeit wohl effizienteste Programm zur Generierung sämtlicher stabiler Mengen eines ungerichteten Graphen in /LOUKAKIS-81/. Die Effizienz dieses an sich schon überragenden Verfahrens wird durch die Bereitstellung einer weiteren Abbruchschranke noch gesteigert. Nach der Lösung von GAP wird ein "retiming" /LEISERSON-83/ eingesetzt, um die Gesamtzahl der Verzögerungswerte auf Prozessor-Prozessorverbindungen zu minimieren.

Zuerst folgt eine Übersicht der in diesem Kapitel relevanten Bezeichnungen:

1. Bezeichnungen.

T	Menge der Elementaroperationen
P	Menge der Prozessoren
$\eta : P \longrightarrow N_0^k$	Bereitstellung von Ressourcen
$s \in N_0^k$	Gesamtverfügbarkeiten
$E \subset T \times P$	Mögliche Zuordnungen
(E, Z)	Zuordnungsgraph
$R \subset v(T)$	Zeitliches Nacheinander der Elementaroperationen
$R_d \subset R$	Datentransfers zwischen Elementaroperationen
$W \subset v(P)$	Verschaltung der Prozessoren
W_l	Leitungen mit limitierten Verzögerungswerten
W_f	Leitungen mit festen Verzögerungswerten
$\delta : P \longrightarrow]0, 1]$	Geschwindigkeiten
$r_p : R_d \longrightarrow Z$	Verzögerungswert des Datentransfers
$\xi : W \longrightarrow N_0$	Verzögerungswert bzw Schranke des Verzögerungswerts
$A_Z^+(S) : S \subset E$	Sämtliche Nachbarn von S in (E, Z) zusammen mit S

S^+	stabile Menge
S^-	verbotene Knotenmenge (backtracked vertices)
$S^{\sim}$	Kandidantenmenge zur Vergrößerung von S^+
$S^\wedge$	Kandidantenmenge zur Vergrößerung von S^+ , die zusätzlich die Gesamtressourcen beachtet.

Gegeben seien T und der Zuordnungsgraph (E, Z) aus (9.2). Es sei $\text{pr}_T : E \longrightarrow T$ die Projektion von E auf T , d.h. $\text{pr}_T(x, p) := x$. Genauso ist $\text{pr}_P : E \longrightarrow P$ die Projektion von E auf P , d.h. $\text{pr}_P(x, p) := p$.

Die Bezeichnungen S^+ , S^- , $S^{\sim}$ sind wie in /LOUKAKIS-81/ gewählt. Zusätzlich zu den in /LOUKAKIS-81/ genannten Abbruchregeln wird nun eine weitere Abbruchregel bewiesen. Sie ergibt sich für die Anwendung des Algorithmus LTMIS aus /LOUKAKIS-81/ zur Lösung von GAP.

2. Lemma.

Es sei S^+ in E stabil und $S^- := E \setminus (A_Z^+(S^+) \cup S^-)$. Es sei $P^+ := \text{pr}_P(S^+)$.

$$P^{\sim} := \{ p \in P \setminus P^+ : \eta(p) + \sum_{q \in P^+} \eta(q) \leq s \}$$

$$S^\wedge := S^{\sim} \cap \text{pr}_P^{-1}(P^{\sim} \cup P^+)$$

Falls $\text{pr}_T(S^+ \cup S^-)$ eine echte Teilmenge von T ist, dann gibt es keine stabile Menge S , für die gilt:

$$(i) \quad \sum_{p \in \text{pr}_P(S)} \eta(p) \leq s$$

$$(ii) \quad S / = T /$$

$$(iii) \quad S \cap S^- = \emptyset$$

$$(iv) \quad S^+ \subset S$$

Beweis:

Ann.: Es gibt eine stabile Menge S mit den Eigenschaften (i)-(iv).

Definiere $S' := S \setminus S^+$.

(iii) und S' stabil zu S^+ liefern: $S' \subset E \setminus (A_Z^+(S^+) \cup S) = S^\sim$.

S' ist sogar eine Teilmenge von $S^\wedge$, denn:

Ann.: $S' \cap (S^\sim \setminus S^\wedge) \neq \emptyset$.

Wähle $(x,p) \in S' \cap (S^\sim \setminus S^\wedge) \Rightarrow p \notin P^\sim \cup P^+ \Rightarrow$

$$\exists l \in \{1, \dots, k\} : \eta(p)_l + \sum_{q \in P^+} \eta(q)_l > s_l$$

Damit gilt:

$$\sum_{p \in \text{pr}_P(S)} \eta(q)_l \geq \eta(p)_l + \sum_{p \in \text{pr}_P(S^+)} \eta(q)_l = \eta(p)_l + \sum_{p \in P^+} \eta(q)_l > s_l$$

Die erste Ungleichung gilt, da $\{(x,p)\} \cup S^+ \subset S$ und $(x,p) \notin S^+$. Die mittlere Gleichung gilt nach Definition von P^+ .

Dies ist jedoch ein Widerspruch zu (i).

Somit gilt: $S' \subset S^\wedge$.

Für jede stabile Menge S von (E, Z) gilt: $|S| = |\text{pr}_T(S)|$. D.h.: pr_T ist auf S injektiv.

Begründung: Ann.: $\exists x \in T : |S \cap \text{pr}_T^{-1}(x)| \geq 2 \Rightarrow$

$\exists p \neq q \in P : (x,p), (x,q) \in S$. Dies ist jedoch ein Widerspruch zur Stabilität von S , da nach Definition von Z aus (9.2) gilt: $[(x,p), (x,q)] \in Z$.

Diese Teilaussage wird nun auf S angewandt:

$$|T| = (* (ii) *) |S| = |\text{pr}_T(S)| = |\text{pr}_T(S^+ \cup S')| \leq |\text{pr}_T(S^+ \cup S^\wedge)| < |T|$$

Die letzte Ungleichung gilt nach Voraussetzung.

Damit ist die Annahme zum Widerspruch geführt.

qed.

Es sei nun S eine stabile Menge in (E, Z) mit (i), (ii) aus Lemma (2). Dann liefert S in natürlicher Weise eine zulässige Lösung $a : E \rightarrow \{0, 1\}$ von GAP mit

$$a(x,p) := \begin{cases} 1 & \text{für } (x,p) \in S \\ 0 & \text{sonst} \end{cases}$$

Begründung: S ist eine $|T|$ -elementige stabile Menge. Damit induziert sie nach (9.3) eine zulässige Lösung von LAP, die zusätzlich noch die Gesamtressourcenre-

striktionen beachtet. Letzteres erkennt man sofort wenn man sich mit (7.4) vergewärtigt, daß gilt:

$$S = a^{-1}(1).$$

Mit diesen Überlegungen liegt es nahe, auch die /T/-elementige stabile Menge S , die (i) aus (2) erfüllt, als zulässige Lösung von GAP zu bezeichnen.

Für den Rest des Kapitels nehmen wir an, daß eine optimale bzw. zulässige Lösung S von GAP gefunden wurde. $P^+ := \text{prp}(S)$ gibt die Menge der benützten Prozessoren an. Es wird nun der von P^+ in (P, W) induzierte Untergraph (P^+, W^+) betrachtet.

Definiere: $W^+_l := W^+ \cap W_l$ und $W^+_f := W^+ \cap W_f$. Die Mengen W^+_l und W^+_f stellen eine Zerlegung der Menge W^+ dar.

Die Leitungen W^+_l stellen Verbindungen mit limitiertem Verzögerungswert dar. Für diese Leitungen wird nun der minimale aktuelle Verzögerungswert $\xi_{\min}$ der Daten, die über diese Leitung geschickt werden, ermittelt:

$$\xi_{\min}: W^+_l \longrightarrow N_0 \quad \xi_{\min}(p, q) := \min \{r_p(x, y) : (x, y) \in R_d \cap (\text{prp}^{-1}(p) \times \text{prp}^{-1}(q))\}$$

Nun wird der gerichtete Graph (P^+, W^+) mit der Pfeilbewertung $w: W \longrightarrow N_0$ betrachtet:

$$w(p, q) := \begin{cases} \xi(p, q) & \text{für } (p, q) \in W^+_f \\ \xi_{\min}(p, q) & \text{für } (p, q) \in W^+_l \end{cases}$$

Für diesen Graphen kann nun ein Retiming /LEISERSON-83/ durchgeführt werden, um die Gesamtsumme der Verzögerungswerte zu minimieren. Auf die Darstellung der algorithmischen Details wird hier verzichtet, weil im Rahmen dieser Arbeit in diesem Teilgebiet keine neuen Erkenntnisse gewonnen wurden. Im folgenden wird das eigentliche Optimierungsproblem formuliert und (dies ist bei /LEISERSON-83/ nur spärlich ausgeführt) das duale Problem ermittelt.

Ein Retiming ist eine Abbildung $rt: P^+ \longrightarrow \mathbb{Z}$. Die Pfeilbewertung w_{rt} eines "retimed" Graphen ergibt sich aus:

$$w_{rt}: W^+ \longrightarrow N_0 \quad w_{rt}(p, q) := w(p, q) - rt(p) + rt(q)$$

Damit ein Retiming zulässig ist, muß die Nichtnegativität von w_{rt} gewährleistet sein. Ein Retiming verändert die Funktionsweise eines synchronen Systems nicht /LEISERSON-83/.

Für jeden Knoten $p \in P^+$ wird nun noch der Eingangsgrad d_{in} und der Ausgangsgrad d_{out} definiert:

$$\begin{aligned} d_{in} : P^+ &\longrightarrow N_0 & d_{in}(p) &:= /V_{W^+}(p)/ \\ d_{out} : P^+ &\longrightarrow N_0 & d_{out}(p) &:= /N_{W^+}(p)/ \end{aligned}$$

Damit kann die Verzögerungsoptimierung formuliert werden:

3. Verzögerungsminimierung (DM- delay minimization)

Gesucht ist eine Abbildung $rt: P^+ \longrightarrow \mathbb{Z}$, sodaß

$$\sum_{p \in P^+} (d_{in}(p) - d_{out}(p)) * rt(p) \longrightarrow \min$$

unter folgender Nebenbedingung:

$$(a) \quad \forall (p, q) \in W^+ : rt(p) - rt(q) \leq w(p, q) \quad \text{enddef.}$$

Dies ist ein lineares Programm, wobei die Variablen keiner Vorzeichenbeschränkung unterliegen. Nun folgt das zu (3) duale Programm:

4. Dualproblem.

Gesucht ist eine Abbildung $\psi : W^+ \longrightarrow \mathbb{R}$, sodaß

$$\sum_{(p, q) \in W^+} w(p, q) * \psi(p, q) \longrightarrow \max$$

unter den Nebenbedingungen:

$$(a) \quad \forall p \in P^+ : \sum_{q \in N_{W^+}(p)} \psi(p, q) - \sum_{q \in V_{W^+}(p)} \psi(q, p) = d_{in}(p) - d_{out}(p)$$

$$(b) \quad \forall (p,q) \in W^+ : \psi(p,q) \geq 0$$

enddef.

Dies ist ein Flußproblem mit inhomogener Flußbedingung. Es kann mit Standardalgorithmen /DOMSCHKE-81/ gelöst werden, wobei die geänderte Flußbedingung (falls $d_{in}(p) - d_{out}(p) \neq 0$) zu beachten ist.

Es bleibt noch zu zeigen, daß (3) die Summe der Verzögerungswerte über alle Pfeile W^+ minimiert. Dies ist sehr leicht einzusehen, denn es gilt:

$$\begin{aligned} \sum_{(p,q) \in W^+} w_{rt}(p,q) &= \sum_{(p,q) \in W^+} w(p,q) - rt(p) + rt(q) = \\ &= \sum_{(p,q) \in W^+} w(p,q) - \sum_{p \in P^+} \sum_{q \in N(p)} rt(p) + \sum_{q \in P^+} \sum_{p \in V(q)} rt(q) = \\ &= \sum_{(p,q) \in W^+} w(p,q) - \sum_{p \in P^+} (/N(p)/ * rt(p) + /V(q)/ * rt(p)) = \\ &= \sum_{(p,q) \in W^+} w(p,q) + \sum_{p \in P^+} (d_{in}(p) * rt(p) - d_{out}(p) * rt(p)) \end{aligned}$$

Der erste Summand ist konstant. Daher wird der zweite Summand minimiert.

11. Neue Anfangszeitpunkte

Dieses Kapitel ist dem unangenehmen Fall gewidmet, daß entweder keine Lösung von GAP existiert oder die gefundene nicht den Erwartungen entspricht. Der zweite Fall tritt beispielweise in folgenden Situationen auf:

- Das ermittelte Verbindungsschema ist nicht in zwei Verdrahtungsebenen zu realisieren.
- Die Ausnutzung einer teuren Resource (z.B. floatingpoint Prozessor, Busanschluß) ist zu gering
- Die generierte Architektur stimmt nicht mit der nachzubildenden Architektur überein.

In diesem Kapitel wird nun "das Pferd von hinten aufgezäumt". Dies bedeutet, daß zuerst ein vereinfachtes Anordnungsproblem SAP gelöst wird. Mit einer Lösung von SAP wird ein Schedulingproblem formuliert. Die Lösung von SAP erfolgt wiederum mit LTMIS aus /LOUKAKIS-81/. Das anschließende Planungsproblem kann auf RCS-Gestalt (4.4) gebracht werden. Die Lösung erfolgt dann mit einem branch&bound Verfahren unter Zuhilfenahme der Ergebnisse von Kapitel 5 und 6.

1. Bezeichnungen.

T	Menge der Elementaroperationen
$R \subset v(T)$	Zeitliches Nacheinander der Elementaroperationen
$R_d \subset R$	Datentransfers zwischen Elementaroperationen
$\tau: T \longrightarrow \mathbb{N}$	Länge des Ausführungsintervalls
$t_{\max}$	Maximaler Ausführungsbeginn
$t: T \longrightarrow \mathbb{N}$	Anfangszeitpunkte
$\Phi = \{\phi_1, \dots, \phi_k\}$	Menge der Ressourcen
$\mu: T \longrightarrow \mathbb{N}_0^k$	Ressourcenanforderungen
$s \in \mathbb{N}_0^k$	Gesamtverfügbarkeiten
P	Menge der Prozessoren
$W \subset v(P)$	Verschaltung der Prozessoren
W_l	Leitungen mit limitiertem Verzögerungswert

W_f	Leitungen mit festem Verzögerungswert
$\eta : P \longrightarrow N_0^k$	Bereitstellung von Ressourcen
$\delta : P \longrightarrow]0,1]$	Geschwindigkeiten
$r_p : R_d \longrightarrow \mathbf{Z}$	Verzögerungswert des Datentransfers
$\xi : W \longrightarrow N_0$	Verzögerungswert(schranke)
$E \subset T \times P$	Mögliche Zuordnungen
$\text{prp} : E \longrightarrow P$	Projektion von E auf P
$I_p : p \in P$	Inkompatibilitäten

Es wird nun ein einfaches Anordnungsproblem formuliert. Die Problemstellung ist unabhängig von den Anfangszeitpunkten der Elementaroperationen. Nur die verfügbaren Ressourcen und die vorgegebene Verdrahtung der Prozessoren spielt nun eine Rolle.

2. Einfaches Anordnungsproblem. (SAP - simple allocation problem)

Definiere die durch die Ressourcenanforderungen bzw -bereitstellungen möglichen Zuordnungen E_Φ von Elementaroperationen an Prozessoren:

$$E_\Phi := \{ (x,p) \in T \times P : \mu(x) \leq \eta(p) \leq s \}$$

Gegeben ist weiterhin eine Kostenabbildung $c : E_\Phi \longrightarrow \mathbf{R}$.

Gesucht ist eine Abbildung $a : E_\Phi \longrightarrow \{0,1\}$, sodaß

$$cTa := \sum_{(x,p) \in E_\Phi} c(x,p) * a(x,p) \longrightarrow \min$$

unter den Nebenbedingungen:

$$(a) \quad \forall x \in T \quad \sum_{p \in N_{E_\Phi}(x)} a(x,p) = 1$$

$$(b) \quad \forall (x,y) \in R_d \quad \forall (p,q) \in v(P) \setminus W : \quad (x,p), (y,q) \in E_\Phi \Rightarrow a(x,p) + a(y,q) \leq 1$$

$$(c) \quad \sum_{p \in \text{prp}(a^{-1}(1))} \eta(p) \leq s$$

enddef.

Es wird nun ein einfacher Zuordnungsgraph (E_Φ, Z_s) definiert:

$$Z_s := \bigcup_{x \in T} \{[(x,p),(x,q)] : (x,p),(x,q) \in E_\Phi\} \cup \\ \text{sym} \{[(x,p),(y,q)] : (x,p),(y,q) \in E_\Phi \text{ und } (x,y) \in R_d \text{ und } (p,q) \in v(P) \setminus W\}$$

Analog zu (9.3) gilt folgender

3. Satz.

Eine Optimallösung von SAP existiert genau dann, wenn es eine stabile Menge S in (E_Φ, Z_s) gibt mit:

$$(i) \quad |S| = |T|$$

$$(ii) \quad \sum_{p \in \text{pr}_P(S)} \eta(p) \leq s$$

Beweis: analog zu (9.3)

Somit kann SAP ebenfalls mit LTMIS unter Benützung von (10.2) gelöst werden. Eine zulässige Lösung $a : E_\Phi \longrightarrow \{0,1\}$ bzw. $S := a^{-1}(1)$ zeigt den Ort jedoch nicht die Zeit der Ausführung von Elementaroperationen. Falls zwei Elementaroperationen am selben Ort ausgeführt werden, müssen sie zeitlich versetzt ablaufen.

Vorerst definiere $a^* : T \longrightarrow P$ mit $a^*(x) = p : \Leftrightarrow a(x,p) = 1$.

Da a für SAP zulässig ist, erfüllt a insbesondere die Nebenbedingung (a) von SAP:

$$\forall x \in T : \exists_1 p \in P : a(x,p) = 1$$

Somit ist a^* wohldefiniert.

Nun folgt die Formulierung eines ressourcenbeschränkten Planungsproblems: Die Ressourcenmenge ist nun P . Für alle Prozessoren $p \in P$ dürfen sich die Operationen $a^*-l(p)$ nicht zeitlich überlappen.

Definiere nun eine Ressourcenanforderung $\mu' : T \longrightarrow N_0^{P/}$ mit

$$\mu'(x)_p := \begin{cases} 1 & a(x,p) = 1 \\ 0 & \text{sonst} \end{cases}$$

Es wird auch eine Funktion $\tau' : T \longrightarrow \mathbb{N}$ mit $\tau'(x) := \lceil \tau(x) * \delta(a^*(x)) \rceil$ definiert.

Die Funktion μ' fordert den Prozessor an, auf den sie gemäß zulässiger Lösung a von SAP implementiert ist. Die Funktion τ gibt die Länge des Ausführungsintervalls für eine Implementierung von x auf $a^*(x)$ an.

Damit ergibt sich folgende

4. Problemformulierung.

Gegeben sei $T, R, R_d, \tau', \mu', t_{\max}, W_f, W_l, \xi$.

Gesucht ist eine Abbildung $t : T \longrightarrow \mathbb{N}$, sodaß

$$\max \{t(x) + \tau(x) : x \in T\} \longrightarrow \min$$

unter folgenden Nebenbedingungen:

- (a) $\forall (x,y) \in R : \quad t(x) + \tau'(x) \leq t(y)$
- (b) $\forall (x,y) \in R_d : \quad \begin{aligned} (a^*(x), a^*(y)) \in W_f &\Rightarrow t(x) + \tau'(x) + \xi(a^*(x), a^*(y)) = t(y) \\ (a^*(x), a^*(y)) \in W_l &\Rightarrow t(x) + \tau'(x) + \xi(a^*(x), a^*(y)) \geq t(y) \end{aligned}$
- (c) $\forall \alpha = 1, \dots, t_{\max} : \quad T_\alpha := \{x \in T : t(x) \leq \alpha < t(x) + \tau'(x)\}$
 $\forall \alpha = 1, \dots, t_{\max} : \quad \sum_{x \in T_\alpha} \mu'(x) \leq 1 \quad (\text{komponentenweise})$
- (d) $\forall x \in T : \quad t(x) \leq t_{\max}$

enddef.

Die Prozessoren sind die verschiedenen Ressourcenklassen (in jeder Klasse ist genau ein Objekt; d.h. $s = 1$). Die Nebenbedingung (c) besagt, daß zu jedem Zeitpunkt α aus den Mengen $a^{*-1}(p)$ jeweils höchstens eine Elementaroperation ausgeführt werden darf. Die Nebenbedingungen (b) ergeben sich aus der Forderung nach Einhaltung der Verzögerungswerte.

Nun wird gezeigt, daß (4) als RCS-Problem umformuliert werden kann:

Die Ordnung R zerfällt in drei Teilmengen:

$$R_1 := \{(x, y) \in R_d : (a^*(x), a^*(y)) \in W_f\}$$

$$R_2 := \{(x, y) \in R_d : (a^*(x), a^*(y)) \in W_1\}$$

$$R_3 := R \setminus (R_1 \cup R_2)$$

Nun werden die Funktionen $\kappa, \lambda : R \rightarrow \mathbb{N}$ festgelegt:

$$\lambda(x, y) := \begin{cases} \tau'(x) & \text{für } (x, y) \in R_3 \\ \tau'(x) & \text{für } (x, y) \in R_2 \\ \tau'(x) + \xi(a^*(x), a^*(y)) & \text{für } (x, y) \in R_1 \end{cases}$$

$$\kappa(x, y) := \begin{cases} \infty & \text{für } (x, y) \in R_3 \\ \tau'(x) + \xi(a^*(x), a^*(y)) & \text{für } (x, y) \in R_2 \\ \tau'(x) + \xi(a^*(x), a^*(y)) & \text{für } (x, y) \in R_1 \end{cases}$$

Mit dieser Festlegung ist eine RCS-Aufgabe definiert worden. Sie ist wohldefiniert, da gilt:

$$\forall x \in T: \tau'(x) \leq \lambda(x, y)$$

Somit kann die Problemformulierung (4) mit einem Lösungsalgorithmus für RCS (approximativ) gelöst werden.

12. Modellierung von Algorithmen

In diesem Kapitel wird ein Modell für Algorithmen entwickelt. Zuerst wird eine PASCAL ähnliche Sprache L festgelegt und daran anschließend eine Normalform für Algorithmen in dieser Sprache definiert. Aus Programmen in dieser Normalform können dann Kausalstrukturen über Elementaroperationen abgeleitet werden. Weiters wird die Wahl eines Prototyps der Kausalstruktur beschrieben, der die zu lösenden RCS-Probleme erheblich verkleinert.

In L lassen sich (in der Regel) auch Assemblerprogramme bzw. Microcodeprogramme formulieren. In dieser Sprache muß jedoch auf explizite GOTO Anweisungen verzichtet werden.

1. Programmiersprache. (L)

Die Programmiersprache L ist ein eingeschränktes PASCAL. Folgende Einschränkungen gegenüber Standard-PASCAL sind maßgebend:

- a) Verzicht auf ein explizites GOTO statement.
- b) Nur einfache Datentypen (real, integer, char, string, boolean) und mehrdimensionale Felder bestehend aus diesen einfachen Datentypen sind zugelassen.
- c) Prozeduren, die im Hauptprogramm aufgerufen werden, benutzen formale Parameter, die sowohl vom Typ "value" als auch vom Typ "variable" sind. Der Rumpf dieser Prozeduren benutzt keine globalen Variablen und ist nicht rekursiv. Damit sind Seiteneffekte bei diesen Prozeduren ausgeschlossen. Für alle anderen Prozeduren bestehen keine Einschränkungen.
- d) Funktionen, die im Hauptprogramm aufgerufen werden, sind ausschließlich vom Typ boolean. Sie weisen nur formale Parameter vom Typ "value" auf. Für den Rumpf solcher Funktionen gelten die selben Einschränkungen wie für Prozeduren, die im Hauptprogramm aufgerufen werden. Für alle übrigen Funktionen gelten keine Einschränkungen.
- e) Die logische Bedingung in einer strukturierten if-Anweisung im Hauptprogramm ist eine Funktion vom Typ boolean.
- f) Schleifen im Hauptprogramm haben stets ein FOR - TO Schema.

- g) Folgende PASCAL Konstrukte sind im Hauptprogramm verboten: CASE, WHILE, REPEAT-UNTIL.

Die Programmiersprache L verbietet damit rekursive Prozeduren, die im Hauptprogramm aufgerufen werden. Die Funktionen, die im Hauptprogramm aufgerufen werden, dienen ausschließlich zur Ablaufsteuerung in if-Anweisungen. Explizite GOTO Anweisungen sind nicht vorgesehen. GOTO Operationen zerfallen nach /ALLEN-83/ in 3 Klassen:

branch GOTO: Hier werden innerhalb eines Blockes Anweisungen übersprungen und *nach* der GOTO Anweisung wieder aufgesetzt.

exit GOTO: Hier wird aus einem Block heraus auf eine *nachfolgende* Anweisung verzweigt.

backward GOTO: Hier wird auf eine *vorangegangene* Anweisung verzweigt.

/ALLEN-83/ zeigt, daß die ersten beiden GOTO's mit strukturierten if-Anweisungen simuliert werden können. Ein backward GOTO jedoch erzeugt implizit eine Schleife.

2. Algorithmen.

Im Rahmen dieser Arbeit sind Algorithmen als Programme in der Programmiersprache L vorgegeben.

Es folgt nun die Definition eines Basisblocks. Diese ist jedoch von dem Blockbegriff gemäß PASCAL zu unterscheiden.

3. Basisblock.

Ein Basisblock ist ein Programmstück, das als fixiertes und geladenes Programm stets über genau einen Befehlszählerstand begonnen und über genau eine Befehlszählerstand beendet wird.

In einem Basisblock gibt es damit nur Verzweigungen innerhalb des Basisblocks und nicht nach außen. Die Wahlmöglichkeiten für einen Basisblock sind sehr vielfältig. So ist beispielsweise das ganze Programm ein Basisblock, genauso wie eine einzelne Anweisung. Für eine Laufschleife ist entweder die gesamte Laufschleife oder ein Teil des Schleifenrumpfes ein Basisblock. Falls innerhalb eines Programms ein Basisblock festgelegt wurde, kann er als Rumpf einer neu zu definierenden Prozedur dienen. Im Programmtext wird dann der Basisblock durch einen Prozeduraufruf ersetzt.

4. Elementaroperationen.

Jedes Programmstück des Hauptprogramms, aus dem ein Prozedurrumpf gebildet werden kann, kann eine Elementaroperation definieren, falls durch die Ersetzung des Programmstücks die Semantik erhalten bleibt. Eine Elementaroperation wird mit dem entsprechenden Prozeduraufruf im Hauptprogramm identifiziert. Die Prozedurvereinbarung ist die funktionale Beschreibung der zugehörigen Elementaroperation(en). Daher ist jeder Aufruf einer Prozedur im Hauptprogramm eine neue Elementaroperation. Wie der Name schon nahelegt, werden Elementaroperationen als unteilbare Einheiten betrachtet. Da die Operationen als atomar angesehen werden, kann man die zugehörigen Prozedurvereinbarungen als Befehlssatz einer hypothetischen Maschine M betrachten. Deshalb heißen Elementaroperationen auch M -Befehle.

Ein Maschinenbefehl einer realen Maschine benützt bei seiner Ausführung gewisse Systemeinheiten (ALU, Datenregister, Shifter, Bus usw.). Dieser Gedanke wird auf die Elementaroperationen übertragen. Zwar stehen keine realen Systemeinheiten von M zur Verfügung, jedoch kann aus der Funktionalität eines M -Befehls auf gewisse Systemeinheiten geschlossen werden. Beispielsweise benötigt eine Elementaroperation, die eine Integeraddition ausführt, einen Volladdierer oder zumindest einen Shifter.

5. Ressourcen.

Der Befehlssatz von M ist die Gesamtheit der Prozedurvereinbarungen von Ele-

mentaroperationen der zugrundeliegenden Programmtextes. Zur Ausführung dieser Befehle benötigt M bestimmte Systemkomponenten, beispielsweise:

- Addierer
- Shifter
- Multiplizierer
- lokale Register
- Busanschlüsse
- floating point Arithmetik
- Speicherzellen usw.

Mit diesen Funktionselementen und einer geeigneten Steuerung implementiert die hypothetische Maschine M ihren Befehlssatz.

6. Anforderung von Ressourcen.

Die einzelnen Befehle von M benötigen für ihre Implementierung jeweils nur eine Teilmenge der Systemkomponenten von M . Diese Komponenten sind Teil der Ressourcenanforderung eines M -Befehls. In manchen Fällen (VLSI) kommt (eine Schätzung für) die Implementierungsfläche und Verlustleistung der benützten Systemkomponenten hinzu.

Gerade in VLSI-Bereich sind Systemkomponenten typischerweise in Zellbibliotheken vorgegeben. Falls die Elementaroperation sehr einfach ist (z.B. Addition zweier Binärzahlen, logischer Vergleich von Bitmustern, Funktion binärer Variablen usw.), kann man für die Elementaroperation ein reales Layout mit den Zellen der Bibliothek benützen. In vielen Fällen sind Elementaroperationen komplexe Programmstücke, für die nur mit großem Aufwand ein reales Layout in VLSI erstellt werden kann. In diesem Fall ist man auf Schätzungen bzw die Erfahrung eines Entwicklers angewiesen, um benötigte Systemkomponenten auszuwählen und um die Implementierungsfläche und Verlustleistung festzulegen.

7. Ausführungsdauer.

Mit Kenntnis (Schätzung) des Zeitbedarfs der einzelnen Systemkomponenten von M wird über die zahlenmäßige Anforderung von Ressourcen eines M -Befehls sein Zeitbedarf (Länge des Ausführungsintervalls) ermittelt (geschätzt).

Die Festlegung von Elementaroperationen dient in erster Linie dazu, ein Programm in Programmiersprache L auf folgende Normalform zu transformieren:

8. Normalform (N1)

Gegeben sei ein Programmtext in Sprache L. Dieser Programmtext ist genau dann in Normalform N1, wenn gilt:

- Das Hauptprogramm ist ein PASCAL Block ohne Unterblöcke.
- Neben "begin" und "end" besteht das Hauptprogramm nur aus Prozeduraufrufen

Damit ist das ganze Hauptprogramm ein Basisblock aus Elementaroperationen.

Das Ziel der Transformationen am Programmtext besteht darin, auf endlich vielen Elementaroperationen eine Kausalstruktur abzuleiten, die die kausalen Zusammenhänge im Programm widerspiegeln. Zwei Sprachmerkmale müssen hierzu noch beseitigt werden: -

- Laufschleifen
- strukturierte if Anweisungen

9. Entfalten von Laufschleifen.

Ein Laufschleifenrumpf besteht aus ein oder mehreren Anweisungen bzw Prozeduraufrufen. Eine Laufschleife der Länge n wird nun dadurch entfaltet, daß n Kopien des Laufschleifenrumpfes aneinandergesetzt werden, wobei die Laufschleifenvariable explizit alle Werte durchläuft.

```
for i := 3 to 5 do
  begin
    proc1(i-1,i-1);
    proc2(i-1,i);
    proc1(i,i + 1);
  end;
```

diese Lauschleife der Länge 3 wird entfaltet:

```

proc1(1,2);
proc2(2,3);
proc1(3,4);
proc1(2,3);
proc2(3,4);
proc1(4,5);
proc1(3,4);
proc2(4,5);
proc1(5,6);

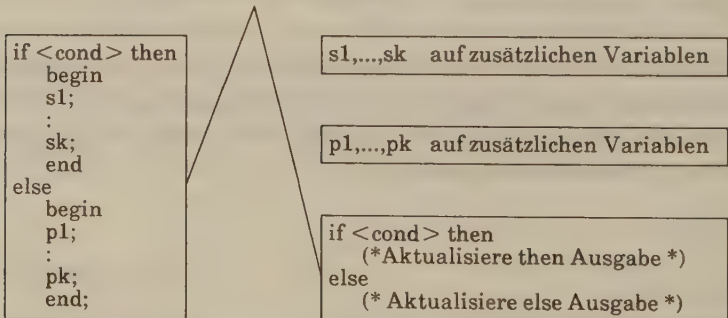
```

Das Entfalten von Lauschleifen ist ein bekanntes Mittel zur Codeoptimierung in Compilern.

10. Entfernen von strukturierten if Anweisungen.

Es bestehen grundsätzlich zwei Möglichkeiten, strukturierte if Anweisungen aus dem Hauptprogramm in N1 zu entfernen. Die boolsche Funktion hat nur Eingabeparameter. Deshalb werden diejenigen Ausgabeparameter der Elementaroperationen der then bzw else Blöcke bestimmt, die im restlichen Hauptprogramm wieder als Parameter auftauchen. Diese Parameter heißen Ausgabeparameter des then- bzw else-Blockes der strukturierten if Anweisung.

neue Elementaroperationen



Die erste Möglichkeit besteht darin, die strukturierte if Anweisung selbst als neue Elementaroperation zu definieren. Die zweite Möglichkeit besteht darin, für die Ausgabeparameter des then- bzw. else-Blockes zusätzliche Variable einzufüh-

ren. Mit diesen neuen Ausgabeparametern wird dann der then **und** else-Block ausgewertet. Anschließend werden abhängig von der boolschen Funktion die wirklichen Ausgabeparameter besetzt. Dieses Updating wird als neue Elementaroperation definiert.

In beiden Fällen wird die if Anweisung aus dem Hauptprogramm in eine Prozedur übertragen.

11. Datenabhängigkeiten (Kausalitäten).

Gegeben sind die Elementaroperationen eines Programmtextes in Normalform N1. Eine Elementaroperation x ist genau dann von einer Elementaroperation y (kausal) abhängig, wenn

- y im Programmtext vor x steht.
- ein Ausgabeparameter a von y ein Eingabeparameter von x ist und alle Operationen z zwischen y und x den Parameter a nicht als Ausgabeparameter enthalten.

12. Bemerkung

In der Normalform N1 sind alle Laufschleifen und strukturierten if-Anweisungen verschwunden. Auf den Elementaroperationen eines Programms in N1 kann eine Kausalstruktur abgeleitet werden, die als Eingabe für RCS dient. Eine letzte Schwierigkeit tut sich auf, falls durch Entfaltung der Laufschleifen sehr viele (> 100) Elementaroperationen entstehen, da der Lösungsaufwand für RCS nicht polynomial beschränkt ist. Falls ein Laufschleifenrumpf aus m Elementaroperationen besteht und n die Länge der Laufschleife ist, so entstehen durch die Entfaltung $n \cdot m$ Elementaroperationen. Es werden nun nicht alle $n \cdot m$ Elementaroperationen betrachtet sondern es wird ein Prototyp festgelegt, für den eine Lösung von RCS ermittelt wird. Die Wahl des Prototyps erfolgt derart, daß die Lösung periodisch auf alle $n \cdot m$ Elementaroperationen fortgesetzt werden kann.

13. Entfaltete Laufschleifen.

Gegeben sei ein Programm in N1, das eine entfaltete Laufschleife enthält. Folgende Bezeichnungen werden gewählt (vergl. Kapitel 4).

- T - alle Elementaroperationen
- T_L - Elementaroperationen der entfalteten Laufschleife
- n - Anzahl Schleifendurchgänge
- m - Anzahl Elementaroperationen im Laufschleifenrumpf
- x_{ij} - i-te Elementaroperation im j-ten Schleifendurchgang
 $i = 1, \dots, m, j = 1, \dots, n$
- R - Kausalitäten über T
- $R(T_L)$ - Kausalitäten innerhalb der entfalteten Laufschleife

Auf $R(T_L)$ wird ein Abstand $\text{dist} : R(T_L) \longrightarrow N_0$ definiert:

$$\text{dist}(x_{ij}, x_{kl}) := l - j$$

$\text{dist}(x, y)$ gibt den Abstand in Iterationen von zwei Elementaroperationen $x, y \in T_L$ mit $(x, y) \in R$ an. Die Abbildung dist ist stets nichtnegativ, da die Operation y nur von einer Operation x anhängen kann, die in der selben oder in einer früheren Iteration liegt.

Mit diesen Vorbereitungen kann der Prototyp definiert werden:

14. Prototyp.

Gegeben sei ein Programm in N1, das eine entfaltete Laufschleife enthält. Gegeben seien weiterhin die Bezeichnungen aus (13). Es sei n_{kgv} das kleinste gemeinsame Vielfache von $\text{dist}(R(T_L)) \setminus \{0\} \subset N$. Falls $\text{dist}(R(T_L)) \setminus \{0\} = \emptyset$, setze:

$$n_{kgv} := 1.$$

Die Wahl eines Prototyps ist nur dann sinnvoll, wenn $2^{*n_{kgv}} \ll n$.

Die ersten n_{kgv} Iterationen der Laufschleife bilden den Hauptteil T_{PH} des Prototyps:

$$T_{PH} := \{x_{ij} : i = 1, \dots, m \text{ und } j = 1, \dots, n_{kgv}\}$$

Die zweiten n_{kgv} Iterationen bilden den Periodenteil T_{pp} des Prototyps:

$$T_{pp} := \{x_{ij} : i = 1, \dots, m \text{ und } j = n_{kgv} + 1, \dots, 2 \cdot n_{kgv}\}$$

Weiters wird eine neue Elementaroperation x_L zusätzlich zu T definiert. Die Operation x_L faßt den Zeitbedarf und die Ressourcenanforderungen der restlichen $n - 2 \cdot n_{kgv}$ Iterationen zusammen.

Der Prototyp T_p wird dann folgendermaßen definiert:

$$T_p := T_{ph} + T_{pp} + \{x_L\}$$

Die Wahl des Prototyps geschieht unter der Annahme, daß nach n_{kgv} Iterationen ein periodisches Verhalten der Laufschleife vorliegt. Dies ist jedoch keineswegs zwingend notwendig. Es kann durchaus der Fall eintreten, daß es nur Lösungen von RCS für das Programm in N_1 gibt, die kein periodisches Verhalten für jeweils n_{kgv} Iterationen aufweisen. Jedoch muß man in der Regel für solch eine Lösung sehr lange Rechenzeiten in Kauf nehmen.

15. Normalisierte entfaltete Laufschleifen.

Gegeben sei eine Programm in N_1 , das eine entfaltete Laufschleife enthält. Gegeben seien die Bezeichnungen aus (13,14) Für einen Prototyp T_p werden nun vier Abbildungen definiert:

$$\text{pos}_H : T_L \longrightarrow T_{ph} \quad \text{pos}_H(x_{ij}) := x_{i,k} \quad \text{mit } k := (j \bmod n_{kgv}) + 1$$

$$\text{pos}_p : T_L \longrightarrow T_{pp} \quad \text{pos}_p(x_{ij}) := x_{i,k} \quad \text{mit } k := (j \bmod n_{kgv}) + 1 + n_{kgv}$$

$$\text{per} : T_L \longrightarrow N_0 \quad \text{per}(x_{ij}) := \lfloor (j-1)/n_{kgv} \rfloor$$

$$\text{pos} : R(T_L) \longrightarrow T \times T$$

$$\text{pos}(x, y) := \begin{cases} (\text{pos}_H(x), \text{pos}_H(y)) & \text{falls } \text{per}(x) = \text{per}(y) \\ (\text{pos}_H(x), \text{pos}_p(y)) & \text{sonst} \end{cases}$$

Eine entfaltete Laufscheife heit genau dann normalisiert, wenn gilt:

- (i) $\forall (x,y) \in R(T_L) : \text{pos}(x,y) \in R$
- (ii) $\forall (x,y) \in R(T_L) : \lambda(x,y) = \lambda(\text{pos}(x,y))$
- (iii) $\forall (x,y) \in R(T_L) : \kappa(x,y) = \kappa(\text{pos}(x,y))$
- (iv) $\forall x \in T_L : \tau(x) = \tau(\text{pos}_H(x))$
- (v) $\forall x \in T_L : \mu(x) = \mu(\text{pos}_H(x))$

Es gilt folgende elementare Eigenschaft:

$$\forall \alpha, \beta \in \{H, P\} \quad \forall x \in T_L : \text{pos}_\alpha(\text{pos}_\beta(x)) = \text{pos}_\alpha(x)$$

16. Prototyp - RCS (P-RCS)

Gegeben sei RCS fr ein Programm in N1, das eine entfaltete Laufscheife enthlt. Gegeben seien weiterhin die Bezeichnungen aus (13,14,15) sowie $n \in \mathbb{N}$ und $\mu^* \in \mathbb{N}_0^k$, soda gilt:

- (i) $\forall x \in T_L \setminus T_{PH} \quad \forall y \in V_R(x) : \text{per}(x) * n > \lambda(y,x)$
- (ii) $\forall x \in T_{PH} : n \geq \tau(x)$

Folgende Bezeichnungen werden gewhlt:

T' - Menge der Elementaroperationen von P-RCS.

$$T' := (T \setminus T_L) \cup T_P.$$

R' - Menge der Kausalitten fr P-RCS.

$$R_1 := \{(x,y) \in T_{PH} \times (T \setminus T_L) : \exists z \in \text{pos}_H^{-1}(x) : y \in V_R(z) \cup N_R(z)\}$$

$$R_2 := \{(x, \text{posp}(x)) : x \in T_{PH}\}$$

$$R_3 := \{(x, x_L) : x \in T_{PH}\}$$

$$R' := R(T') \cup R_1 \cup R_2 \cup R_3.$$

$t'_{\max}$ - spätester Ausführungsbeginn für alle Elementaroperationen aus T' .

$$t'_{\max} := t_{\max} - \lfloor (n-1)/n_{\text{kgv}} \rfloor * \pi$$

λ', κ' - minimaler bzw maximaler Zeitabstand von $x, y \in T'$ mit $(x, y) \in R'$

Für alle $(x, y) \in T_{PH} \times (T \setminus T_L)$ definiere:

$$L_1(x, y) := \{ \text{per}(z) * \pi - \kappa(y, z) : \text{pos}_H(z) = x \wedge y \in V_R(z) \wedge z \in T_L \setminus T_{PH} \} \cup \{ \tau(x) \}$$

$$L_2(x, y) := \{ \text{per}(z) * \pi + \lambda(z, y) : \text{pos}_H(z) = x \wedge y \in N_R(z) \wedge z \in T_L \}$$

$$K_1(x, y) := \{ \text{per}(z) * \pi - \lambda(y, z) : \text{pos}_H(z) = x \wedge y \in V_R(z) \wedge z \in T_L \setminus T_{PH} \}$$

$$K_2(x, y) := \{ \text{per}(z) * \pi + \kappa(z, y) : \text{pos}_H(z) = x \wedge y \in N_R(z) \wedge z \in T_L \}$$

$$\lambda'(x, y) := \begin{cases} \max(L_1(x, y) \cup L_2(x, y)) & \text{falls } (x, y) \in T_{PH} \times (T \setminus T_L) \cap R' \\ \pi & \text{falls } (x, y) \in R_2 \\ \tau(x) & \text{falls } (x, y) \in R_3 \\ \lambda(x, y) & \text{sonst} \end{cases}$$

$$\kappa'(x, y) := \begin{cases} \min(K_1(x, y) \cup K_2(x, y) \cup \{t'_{\max}\}) & \text{falls } (x, y) \in T_{PH} \times (T \setminus T_L) \cap R' \\ \pi & \text{falls } (x, y) \in R_2 \\ 2 * \pi & \text{falls } (x, y) \in R_3 \\ \kappa(x, y) & \text{sonst} \end{cases}$$

Die Abbildungen τ bzw μ werden von T auf $T \cup \{x_L\}$ fortgesetzt durch:

$$\tau(x_L) := \lfloor (n-1)/n_{\text{kgv}} \rfloor * \pi$$

$$\mu(x_L) := \mu^*$$

Damit ist P-RCS durch $T', R', \tau, \Phi, \mu, s, \lambda', \kappa', t'_{\max}$ gemäß (4.4) festgelegt.

Die Bedingung (i) dient dazu sicherzustellen, daß gilt:

$$\forall (x, y) \in T_{PH} \times (T \setminus T_L) \cap R' : (K_1(x, y) \neq \emptyset \Rightarrow \min K_1(x, y) > 0)$$

und somit die Positivität von κ' gesichert ist.

Es folgt nun der Höhepunkt dieses Kapitels: Es wird gezeigt, daß aus einer Lösung von P-RCS eine periodische Lösung von RCS gewonnen werden kann. Da für alle x aus T_{PH} $\lambda'(x, \text{posp}(x)) = \pi$ gilt, dient die Bedingung (ii) dazu $\tau(x) \leq \lambda'(x, \text{posp}(x))$ gemäß (4.3) sicherzustellen.-

17. Satz.

Gegeben sei RCS für ein Programm in N_1 , das eine entfaltete normalisierte Laufschleife enthält. Gegeben seien weiterhin die Bezeichnungen von (13,...,16).

Es sei $\pi \in \mathbb{N}$, sodaß gilt:

- (i) $\forall x \in T_L \setminus T_{PH} \quad \forall y \in V_R(x) : \text{per}(x) * \pi > \lambda(y, x)$
- (ii) $\forall x \in T_{PH} : \pi \geq \tau(x)$
- (iii) $\forall (x, y) \in R \cap R_2 : \lambda(x, y) \leq \pi \leq \kappa(x, y)$

Sei $\mu^* \in \mathbb{N}_0^k$, sodaß mit $R^* := t(R)(T_L) \cup \{(x, y) \in v(T_L) : \text{pos}_H(x) = \text{pos}_H(y)\}$ gilt:

$$(iv) \quad \forall S \subset T_L : S \text{ in } R^* \text{ stabil} \quad : \quad \sum_{x \in S} \mu(x) \leq \mu^*$$

Sei $t' : T' \longrightarrow \mathbb{N}$ eine zulässige Lösung von P-RCS. Dann ist $t : T \longrightarrow \mathbb{N}$ mit

$$t(x) := \begin{cases} t'(\text{pos}_H(x)) + \text{per}(x) * \pi & \text{für } x \in T_L \setminus (T_{PH} \cup T_{PP}) \\ t'(x) & \text{sonst} \end{cases}$$

eine zulässige Lösung von RCS.

Beweis:

Vorerst werden einige Hilfsaussagen bewiesen:

Hilfssatz 1:

$$\forall x \in T_{PP} \quad : \quad t(x) = t'(\text{pos}_H(x)) + \pi$$

Begründung:

Sei $x \in T_{PP}$. Es folgt nach Definition von T_{PP} , daß gilt:

$(\text{pos}_H(x), x) = (\text{pos}_H(x), \text{posp}(\text{pos}_H(x))) \in R_2$. Mit Definition von λ' bzw κ' und mit t' zulässig für P-RCS folgt:

$$\begin{aligned} t(x) = t'(x) &\leq t'(\text{pos}_H(x)) + \lambda'(\text{pos}_H(x), x) = t'(\text{pos}_H(x)) + \pi = \\ &t'(\text{pos}_H(x)) + \kappa'(\text{pos}_H(x), x) \leq t'(x) = t(x) \end{aligned}$$

Damit gilt überall die Gleichheit. qed

Hilfssatz 2:

$$\forall x \in T_L \quad : \quad t(x) = t'(\text{pos}_H(x)) + \text{per}(x) * \pi$$

Begründung:

Falls $x \in T_{PH}$, folgt: $\text{per}(x) = 0$ und $\text{pos}_H(x) = x$

Damit gilt: $t'(\text{pos}_H(x)) + \text{per}(x) * \pi = t'(\text{pos}_H(x)) = t'(x) = t(x)$

Falls $x \in T_{PP}$, folgt die Aussage nach Hilfssatz 1.

Falls $x \in T_L \setminus (T_{PH} \cup T_{PP})$, gilt die Aussage nach Definition von t . qed.

Hilfssatz 3:

$$\forall (x, y) \in R \cap (T_L \times (T \setminus T_L)) \quad : \quad \lambda'(\text{pos}_H(x), y) \geq \lambda(x, y) + \text{per}(x) * \pi$$

Begründung:

Falls $x \in T_{PH}$, folgt mit der Definition von λ' :

$$\lambda'(\text{pos}_H(x), y) = \lambda'(x, y) \geq \max L_2(x, y) \geq \lambda(x, y) + \text{per}(x) * \pi$$

Die letzte Ungleichung gilt, da $x = \text{pos}_H(x)$ und $y \in N_R(x)$

Falls $x \in T_L \setminus T_{PH}$, so folgt mit der Definition von $L_2(\text{pos}_H(x), y)$:

$$\text{per}(x) * \pi + \lambda(x, y) \in L_2(\text{pos}_H(x), y)$$

Nach Definition von λ' folgt:

$$\lambda'(\text{pos}_H(x), y) \geq \max L_2(\text{pos}_H(x), y) \geq \lambda(x, y) + \text{per}(x) * \pi \quad \text{qed}$$

Hilfssatz 4:

$$\forall (x, y) \in R \cap (T_L \times (T \setminus T_L)) : \kappa'(\text{pos}_H(x), y) \leq \kappa(x, y) + \text{per}(x) * \pi$$

Begründung:

Falls $x \in T_{PH}$, folgt mit der Definition von κ' :

$$\kappa'(\text{pos}_H(x), y) = \kappa'(x, y) \leq \min K_2(x, y) \leq \kappa(x, y) + \text{per}(x) * \pi$$

Die letzte Ungleichung gilt, da $x = \text{pos}_H(x)$ und $y \in N_R(x)$

Falls $x \in T_L \setminus T_{PH}$, so folgt mit der Definition von $K_2(\text{pos}_H(x), y)$:

$$\text{per}(x) * \pi + \kappa(x, y) \in K_2(\text{pos}_H(x), y)$$

Nach Definition von κ' folgt:

$$\kappa'(\text{pos}_H(x), y) \leq \min K_2(\text{pos}_H(x), y) \leq \kappa(x, y) + \text{per}(x) * \pi \quad \text{qed}$$

Hilfssatz 5:

$$\forall (x, y) \in R(T_L) : \text{per}(x) \leq \text{per}(y) \leq \text{per}(x) + 1$$

Begründung:

Sei $(x, y) \in R(T_L)$ beliebig. Da y nur von einer Operation aus der selben oder einer früheren Iteration abhängen kann, gilt $\text{per}(x) \leq \text{per}(y)$.

$$x = x_{i,j} \text{ und } y = x_{k,l}$$

Nach Definition von n_{kgv} gilt:

$$n_{kgv} \geq d(x, y) = d(x_{i,j}, x_{k,l}) = l - j = l - 1 - (j - 1) \Rightarrow$$

$$1 \geq (l-1)/n_{kgv} - (j-1)/n_{kgv} \Rightarrow$$

$$(j-1)/n_{kgv} + 1 \geq (l-1)/n_{kgv} \Rightarrow$$

$$\lfloor (j-1)/n_{kgv} + 1 \rfloor \geq \lfloor (l-1)/n_{kgv} \rfloor \Rightarrow$$

$$\lfloor (j-1)/n_{kgv} \rfloor + 1 \geq \lfloor (l-1)/n_{kgv} \rfloor \Rightarrow$$

$$\text{per}(x) + 1 \geq \text{per}(y) \quad \text{qed}$$

Nun folgt der Nachweis der Nebenbedingungen (a) - (d) von RCS für t.

Nebenbedingungen (a)/(b):

$$\forall (x,y) \in R \quad : \quad t(x) + \lambda(x,y) \leq t(y) \text{ und } t(x) + \kappa(x,y) \geq t(y)$$

Die folgenden Beweisschritte benutzen folgende Sachverhalte:

- die Zulässigkeit von t' für P-RCS
- die Definitionen für λ' und κ'
- die Hilfsaussagen 1,...,5
- Normalisiertheit der entfalteten Laufschleife

1. Fall: $(x,y) \in R(T_L)$.

1.1. Fall: $\text{per}(x) = \text{per}(y)$

$$t(x) + \lambda(x,y) = t'(\text{pos}_H(x)) + \lambda(\text{pos}(x,y)) + \text{per}(x) * \pi =$$

$$t'(\text{pos}_H(x)) + \lambda(\text{pos}_H(x), \text{pos}_H(y)) + \text{per}(x) * \pi =$$

$$t'(\text{pos}_H(x)) + \lambda'(\text{pos}_H(x), \text{pos}_H(y)) + \text{per}(x) * \pi \leq$$

$$t'(\text{pos}_H(y)) + \text{per}(x) * \pi = t'(\text{pos}_H(y)) + \text{per}(y) * \pi = t(y)$$

$$t(x) + \kappa(x,y) = t'(\text{pos}_H(x)) + \kappa(\text{pos}(x,y)) + \text{per}(x) * \pi =$$

$$t'(\text{pos}_H(x)) + \kappa(\text{pos}_H(x), \text{pos}_H(y)) + \text{per}(x) * \pi =$$

$$t'(\text{pos}_H(x)) + \kappa'(\text{pos}_H(x), \text{pos}_H(y)) + \text{per}(x) * \pi \geq$$

$$t'(\text{pos}_H(y)) + \text{per}(x) * \pi = t'(\text{pos}_H(y)) + \text{per}(y) * \pi = t(y)$$

1.2. Fall: $\text{per}(x) \neq \text{per}(y)$ mit Hilfssatz 5 folgt: $\text{per}(x) + 1 = \text{per}(y)$

1.2.1. Fall: $\text{pos}(x,y) \in R_3$; dies liefert:

$$(\text{pos}_H(x), \text{posp}(y)) \in R \cap (T_{PH} \times T_{PP}) \quad \text{und} \quad \text{pos}_H(x) = \text{pos}_H(y)$$

$$t(x) + \lambda(x,y) = t'(\text{pos}_H(x)) + \lambda(\text{pos}(x,y)) + \text{per}(x) * \pi =$$

$$t'(\text{pos}_H(x)) + \lambda(\text{pos}_H(x), \text{posp}(y)) + \text{per}(x) * \pi \leq \quad (* \text{ iii } *)$$

$$t'(\text{posp}(y)) + \text{per}(x) * \pi = t(\text{posp}(y)) + \text{per}(x) * \pi =$$

$$t'(\text{pos}_H(\text{posp}(y)) + \pi + \text{per}(x) * \pi = t'(\text{pos}_H(y)) + (\text{per}(x) + 1) * \pi = t(y)$$

$$\begin{aligned}
t(x) + \kappa(x,y) &= t'(\text{pos}_H(x)) + \kappa(\text{pos}(x,y)) + \text{per}(x) * n = \\
t'(\text{pos}_H(x)) + \kappa(\text{pos}_H(x), \text{posp}(y)) + \text{per}(x) * n &\geq \quad (* \text{iii} *) \\
t'(\text{posp}(y)) + \text{per}(x) * n &= t(\text{posp}(y)) + \text{per}(x) * n = \\
t'(\text{pos}_H(\text{posp}(y)) + n + \text{per}(x) * n &= t'(\text{pos}_H(y)) + (\text{per}(x) + 1) * n = t(y)
\end{aligned}$$

1.2.2. Fall: $\text{pos}(x,y) \notin R_3$.

$$\begin{aligned}
t(x) + \lambda(x,y) &= t'(\text{pos}_H(x)) + \lambda(\text{pos}(x,y)) + \text{per}(x) * n = \\
t'(\text{pos}_H(x)) + \lambda(\text{pos}_H(x), \text{posp}(y)) + \text{per}(x) * n &= \\
t'(\text{pos}_H(x)) + \lambda'(\text{pos}_H(x), \text{posp}(y)) + \text{per}(x) * n &\leq \\
t'(\text{posp}(y)) + \text{per}(x) * n &= t(\text{posp}(y)) + \text{per}(x) * n = \\
t'(\text{pos}_H(\text{posp}(y)) + n + \text{per}(x) * n &= t'(\text{pos}_H(y)) + (\text{per}(x) + 1) * n = t(y)
\end{aligned}$$

$$\begin{aligned}
t(x) + \kappa(x,y) &= t'(\text{pos}_H(x)) + \kappa(\text{pos}(x,y)) + \text{per}(x) * n = \\
t'(\text{pos}_H(x)) + \kappa(\text{pos}_H(x), \text{posp}(y)) + \text{per}(x) * n &= \\
t'(\text{pos}_H(x)) + \kappa'(\text{pos}_H(x), \text{posp}(y)) + \text{per}(x) * n &\geq \\
t'(\text{posp}(y)) + \text{per}(x) * n &= t(\text{posp}(y)) + \text{per}(x) * n = \\
t'(\text{pos}_H(\text{posp}(y)) + n + \text{per}(x) * n &= t'(\text{pos}_H(y)) + (\text{per}(x) + 1) * n = t(y)
\end{aligned}$$

2. Fall: $(x,y) \in R \cap (T_L \times (T \setminus T_L))$

$$\begin{aligned}
t(x) + \lambda(x,y) &= t'(\text{pos}_H(x)) + \lambda(x,y) + \text{per}(x) * n \leq (* \text{Hilfsaussage 3} *) \\
t'(\text{pos}_H(x)) + \lambda'(\text{pos}_H(x), y) &\leq t'(y) = t(y)
\end{aligned}$$

$$\begin{aligned}
t(x) + \kappa(x,y) &= t'(\text{pos}_H(x)) + \kappa(x,y) + \text{per}(x) * n \geq (* \text{Hilfsaussage 4} *) \\
t'(\text{pos}_H(x)) + \kappa'(\text{pos}_H(x), y) &\geq t'(y) = t(y)
\end{aligned}$$

3. Fall: $(x,y) \in R \cap ((T \setminus T_L) \times T_L)$

3.1. Fall: $y \in T_{PH}$:

$$t(x) + \lambda(x,y) = t'(x) + \lambda'(x,y) \leq t'(y) = t(y)$$

$$t(x) + \kappa(x,y) = t'(x) + \kappa'(x,y) \geq t'(y) = t(y)$$

3.2. Fall: $y \in T_L \setminus T_{PH}$:

Da $x \in V_R(y)$, folgt: $\text{per}(y) * \pi - \lambda(x, y) \in K_1(\text{pos}_H(y), x)$

Damit gilt: $\kappa'(\text{pos}_H(y), x) \leq \min K_1(\text{pos}_H(y), x) \leq \text{per}(y) * \pi - \lambda(x, y)$

Damit gilt insgesamt:

$$t(x) + \lambda(x, y) \leq t'(x) + \text{per}(y) * \pi - \kappa'(\text{pos}_H(y), x) \leq$$

$$t'(x) + \text{per}(y) * \pi + t'(\text{pos}_H(y)) - t'(x) = t(y)$$

Da $x \in V_R(y)$, folgt: $\text{per}(y) * \pi - \kappa(x, y) \in L_1(\text{pos}_H(y), x)$

Damit gilt: $\lambda(\text{pos}_H(y), x) \geq \max L_1(\text{pos}_H(y), x) \geq \text{per}(y) * \pi - \kappa(x, y)$

Damit gilt insgesamt:

$$t(x) + \kappa(x, y) \geq t'(x) + \text{per}(y) * \pi - \lambda'(\text{pos}_H(y), x) \geq$$

$$t'(x) + \text{per}(y) * \pi + t'(\text{pos}_H(y)) - t'(x) = t(y)$$

Damit sind sämtliche Pfeile von R untersucht und die Nebenbedingungen (a) und (b) gezeigt.

Nebenbedingung(c):

$$\forall \alpha = 1, \dots, t_{\max}: \quad T_\alpha := \{x \in T: t(x) \leq \alpha < t(x) + \tau(x)\}$$

$$\forall \alpha = 1, \dots, t_{\max}: \quad \sum_{x \in T} \mu(x) \leq s \quad (\text{komponentenweise})$$

Sei $\alpha \in \{1, \dots, t_{\max}\}$

Zeige zunächst eine

Hilfsaussage 6:

$$\forall x \in U := T_L \setminus (T_{PP} \cup T_{PH}) \quad : \quad t'(x_L) \leq t(x), \quad t(x) + \tau(x) \leq t'(x_L) + \tau(x_L)$$

Begründung:

Sei $x \in U$. Damit kann x frühestens in der $2 * n_{kgv} + 1$ ten Iteration der Laufschleife liegen. Damit gilt: $\text{per}(x) \geq 2$.

Nach Definition von R' gilt: $(\text{pos}_H(x), x_L) \in R'$

Damit gilt mit der Definition von λ' und κ' auf dem Pfeil $(\text{pos}_H(x), x_L)$:

$$t'(x_L) \leq t'(\text{pos}_H(x)) + \kappa'(\text{pos}_H(x), x_L) = t'(\text{pos}_H(x)) + 2 * n \leq \\ t'(\text{pos}_H(x)) + \text{per}(x) * n = t(x)$$

$$t(x) + \tau(x) = t'(\text{pos}_H(x)) + \text{per}(x) * n + \tau(x) \leq \\ t'(x_L) - \lambda'(\text{pos}_H(x), x_L) + \lfloor (n-1)/n_{\text{kgv}} \rfloor * n + \tau(x) = \\ t'(x_L) - \tau(\text{pos}_H(x)) + \tau(x) + \lfloor (n-1)/n_{\text{kgv}} \rfloor * n = \\ t'(x_L) + \lfloor (n-1)/n_{\text{kgv}} \rfloor * n = t'(x_L) + \tau(x_L) \quad \text{qed.}$$

Zeige nun, daß $U \cap T_a$ stabil in $t(R)$ ist.

Begründung: Falls eine Kette zwischen zwei Elementen von $U \cap T_a$ bestünde, würde sie ein zeitliches Nacheinander erzwingen. Die Operationen in T_a überlappen jedoch alle im Zeitpunkt a .

Zeige nun weiters, daß gilt:

$$\forall x, y \in U \cap T_a \quad : \quad (x \neq y \Rightarrow \text{pos}_H(x) \neq \text{pos}_H(y))$$

Begründung:

Ann.: Es gibt $x \neq y$ in $U \cap T_a$, sodaß $\text{pos}_H(x) = \text{pos}_H(y)$

Damit müssen die beiden Elementaroperationen mindestens n_{kgv} (eine Periode) auseinanderliegen. Somit gilt: $|\text{per}(x) - \text{per}(y)| \geq 1$

$$\text{oEdA:} \quad \text{per}(x) \geq \text{per}(y) + 1$$

Damit gilt insgesamt:

$$\alpha < t(y) + \tau(y) = t'(\text{pos}_H(y)) + \text{per}(y) * n + \tau(y) \leq (* \text{ii} *) \\ t'(\text{pos}_H(y)) + \text{per}(y) * n + n = t'(\text{pos}_H(y)) + (\text{per}(y) + 1) * n \leq \\ t'(\text{pos}_H(y)) + \text{per}(x) * n = t'(\text{pos}_H(x)) + \text{per}(x) * n = t(x) \leq \alpha$$

Dies ist ein Widerspruch qed

Damit ist $U \cap T_a$ sogar in R^* stabil.

Definiere nun: $T'_a := \{x \in T' : t'(x) \leq \alpha < t'(x) + \tau(x)\}$

Zeige nun:

$$T'_a \supset \begin{cases} T_a & \text{falls } U \cap T_a = \emptyset \\ (T_a \setminus U) \cup \{x_L\} & \text{sonst} \end{cases}$$

Begründung:

1. Fall: $U \cap T_a = \emptyset$

$$x \in T_a \Rightarrow t(x) \leq a < t(x) + \tau(x) \Rightarrow (* U \cap T_a = \emptyset *)$$

$$t'(x) \leq a < t'(x) + \tau(x) \Rightarrow x \in T'_a$$

2. Fall: $U \cap T_a \neq \emptyset$

Wähle $x \in U \cap T_a$. Mit Hilfsaussage 6 und der Definition von T_a gilt:

$$t'(x_L) \leq t(x) \leq a < t(x) + \tau(x) \leq t'(x_L) + \tau(x_L)$$

Damit ist x_L ein Element von T'_a .

Sei nun $x \in T_a \setminus U$: Da auf $T_a \setminus U$ t identisch mit t' ist, folgt:

$$t'(x) = t(x) \leq a < t(x) + \tau(x) = t'(x) + \tau(x)$$

Damit ist $x \in T'_a$. qed

Damit kann nun die Nebenbedingung (c) gezeigt werden:

1. Fall: $U \cap T_a = \emptyset$:

$$\sum_{x \in T_a} \mu(x) \leq \sum_{x \in T'_a} \mu(x) \leq s$$

2. Fall: $U \cap T_a \neq \emptyset$

$$\sum_{x \in T_a} \mu(x) = \sum_{x \in T_a \cap U} \mu(x) + \sum_{x \in T_a \setminus U} \mu(x) \leq \mu(x_L) + \sum_{x \in T_a \setminus U} \mu(x) \leq \sum_{x \in T'_a} \mu(x) \leq s$$

Die erste Ungleichung gilt, da $U \cap T_a$ stabil in R^* ist, die zweite wegen der Teilmengenrelation und die dritte wegen der Zulässigkeit von t' für P-RCS.

Nun muß noch die Nebenbedingung (d) nachgeprüft werden:

$$\forall x \in T: \quad t(x) \leq t_{\max}$$

Falls $x \in T_L \setminus (T_{PP} \cup T_{PH})$, gilt mit Hilfsaussage 6:

$$t(x) < t(x) + \tau(x) \leq t'(x_L) + \tau(x_L) \leq t'_{\max} + \lfloor (n-1)/n_{\text{kgv}} \rfloor * n = t_{\max}.$$

Andersfalls gilt:

$$t(x) = t'(x) \leq t'_{\max} < t_{\max}$$

Dies beendet den Beweis von Satz 17.

Mit Satz 17 ist es möglich aus der Lösung eines P-RCS eine Lösung von RCS zu konstruieren. Der Vorteil von P-RCS besteht darin, daß $|T'| \ll |T|$. Dies reduziert der erforderlichen algorithmischen Lösungsaufwand beträchtlich.

13. Modellierung paralleler Rechenanlagen

In diesem Kapitel wird ein Modell für parallele Rechenanlagen entwickelt. Folgende Typen von Rechnerarchitekturen werden modelliert:

- Datenpfade von (horizontal mikroprogrammierbare) Prozessoren
- Systolische Architekturen
- SIMD Rechner
- MIMD Rechner

Zuerst wird ein 2-Phasentakt eingeführt. Alle Systemkomponenten paralleler Rechenanlagen sind mit diesem Takt versorgt. Wichtig ist, daß jede Systemkomponente paralleler Rechenanlagen als Prozessor bezeichnet wird. Neben befehlsgesteuerten Prozessoren sind auch direkte Hardwareimplementierungen einfacher Funktionen (add, shift, move,...) als Systemkomponenten zugelassen. In diesem Kapitel wird außerdem ein Prototyp GAP definiert, um die Größe des zu lösenden GAP zu reduzieren.

1. Systemtaktung.

Das parallele Rechensystem wird mit einem 2-Phasen Takt /MEAD-80/ p 65 gesteuert.

Gegeben sind zwei Konstante $\beta_{\min}$ und $\beta_{\text{per}} \in \mathbb{R}^+$.

Gegeben sind weiterhin zwei Funktionen $\chi_j : \mathbb{R} \rightarrow \{0,1\} \quad : j = 1,2$

$$UP(\chi_j) := \{x \in \mathbb{R}^+ : \chi_j(x) = 1 \text{ und } \forall 0 < \varepsilon < \beta_{\min} : \chi_j(x - \varepsilon) = 0 \text{ und } \chi_j(x + \varepsilon) = 1\}$$

$$DO(\chi_j) := \{x \in \mathbb{R}^+ : \chi_j(x) = 1 \text{ und } \forall 0 < \varepsilon < \beta_{\min} : \chi_j(x - \varepsilon) = 1 \text{ und } \chi_j(x + \varepsilon) = 0\}$$

Die Funktionen $(\chi_j)_{j=1,2}$ heißen nichtüberlappender 2-Phasentakt, wenn gilt:

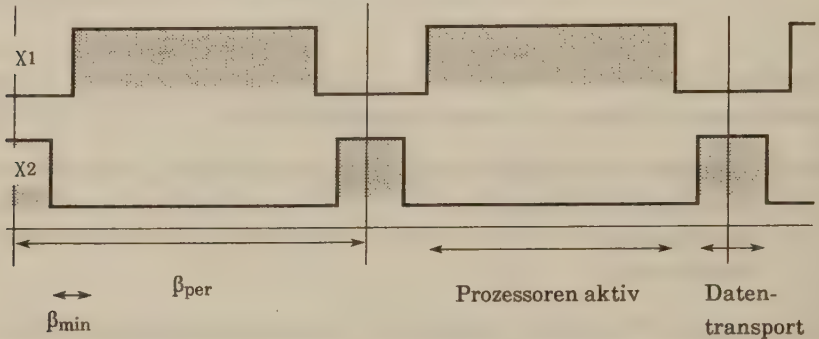
$\exists x_1, x_2, y_2, y_2 \in \mathbb{R} \quad \text{mit } 0 < x_1, x_2, y_2, y_2 < \beta_{\text{per}}$ sodaß gilt:

$$(i) \quad \forall j = 1,2 : \quad UP(\chi_j) = x_j + N_0 * \beta_{\text{per}}$$

$$(ii) \quad \forall j = 1,2 : \quad DO(\chi_j) = y_j + N_0 * \beta_{\text{per}}$$

$$(iii) \quad \forall x \in \mathbb{R} : x_1(x) * x_2(x) = 0$$

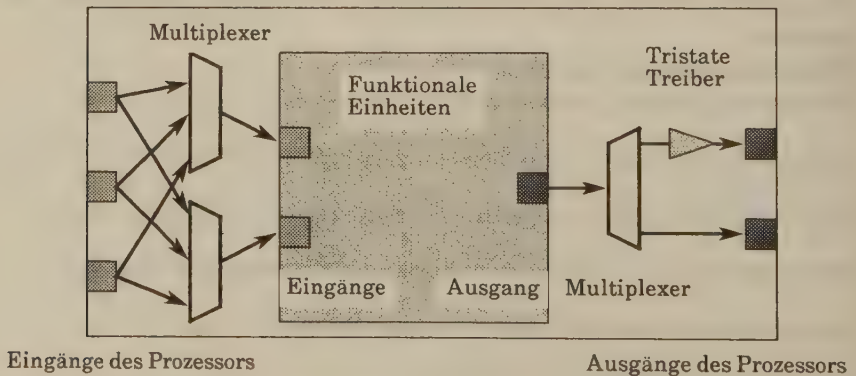
Typischerweise hat ein 2-Phasentakt folgendes Aussehen:



Zu den Zeitpunkten $x_1^{-1}(1)$ arbeiten die Prozessoren während zu $x_2^{-1}(1)$ Daten in Register geschrieben und aus Registern gelesen werden.

2. Prozessoren.

Prozessoren sind entweder befehlsgesteuerte Prozessoren oder direkte Hardwareimplementierungen einer vorgegebenen Funktion. Die Prozessoren verfügen jeweils über mehrere Eingangs- und Ausgangsanschlüsse (I/O ports).



Diese Send- und Empfangsstellen sind über Multiplexer mit den Ausgängen bzw

Eingängen der funktionalen Einheiten verbunden. Diese Multiplexer bestimmen den spezifischen Eingang bzw Ausgang des Prozessors der Eingabe- bzw. Ausgabedaten. Falls ein Ausgang lesend auf einen Bus zugreift, ist diesem Ausgang ein Tristate-Treiber vorangeschaltet.

3. Register.

Die Register sind die Speicherzellen der Architektur. Ein Register verfügt über genau einen Eingang und einen Ausgang, der lesend bzw schreibend auf eine Leitung oder einen Bus zugreift. Mehrere Register können zu Speicherblöcken zusammengefaßt werden. In der Regel hat ein Speicherblock viel weniger Ein- und Ausgänge als Register. Auf die Register eines Speicherblocks kann parallel lesend und schreibend zugegriffen werden, wobei ein paralleles Schreiben nur auf verschiedenen Registern möglich ist. Die Anzahl der gleichzeitigen Lese- und Schreibzugriffe ergibt sich jeweils aus der Anzahl der Ein- und Ausgänge des Speicherblocks. In Registern außerhalb von Speicherblöcken haben die gespeicherten Daten jeweils eine für dieses Register feste Verweildauer (in Systemtakt). Für Register in Speicherblöcken gilt diese Einschränkung nicht. Den Variablen des Hauptprogramms im zugrundeliegenden L Programm wird für eine Lösung von RCS ein Register fest zugeordnet. Dies ist möglich, da die Lebensdauer der einzelnen Variablen mit einer Lösung von RCS festliegt.

4. Datenleitungen und logische Verbindungen.

Die Kommunikation der Prozessoren untereinander erfolgt ausschließlich über Register (einzeln oder in Speicherblöcken). Falls genau ein Ausgang eines Registers oder Speicherblocks mit genau einem Eingang eines Prozessors verbunden ist (oder auch umgekehrt), so wird diese Verbindung als simple Datenleitung realisiert. Andernfalls erfolgt die Realisierung mit einem Bus. Falls ein Ausgang schreibend auf einen Bus zugreift, ist diesem Ausgang ein Treiber vorgeschaltet. Falls auf einen Bus mehrere schreibende Zugriffe möglich sind, müssen in den entsprechenden Ausgängen sogar Tristate-Treiber vorhanden sein.

Ein Prozessor p ist genau dann mit einem Prozessor q logisch verbunden, wenn es ein Register (evt. in einem Speicherblock) gibt, sodaß p schreibend und q lesend auf dieses Register zugreifen kann. Den Programmvariablen im Hauptprogramm des betrachteten L Programms ist jeweils ein Register fest zugeordnet, in der diese Va-

riable abgespeichert wird. Somit sind (streng genommen) die logischen Verbindungen abhängig von den verschiedenen Programmvariablen.

5. Datenpfade von Mikroprozessoren.

Ein (horizontal programmierbarer) Mikroprozessor besteht in der Regel aus einem Mikroprogrammspeicher, einem Steuerwerk und einem Datenpfad. Der Datenpfad /MEAD-80/ p. 166 besteht wiederum aus mehreren Verarbeitungselementen (Shifter, ALU, ...) und mehreren Registern. Sämtliche Komponenten des Datenpfades werden als Prozessoren bezeichnet. Zwischen den einzelnen Prozessoren ist ein Verbindungsnetzwerk vorhanden, auf den der Datentransport stattfindet. Ein Steuerwort eines Mikroprozessors zerfällt in mehrere Bereiche. Die Steuerleitungen der einzelnen Bereiche dienen entweder zur Steuerung genau einer Systemkomponente (Prozessors) oder zur Steuerung des Datentransports.

6. Systolische Architekturen.

Systolische Architekturen bestehen aus einer Vielzahl von Verarbeitungselementen (Prozessoren). Die Menge der Prozessoren kann typischerweise in wenige Teilmengen zerlegt werden, wobei die Prozessoren in den einzelnen Teilmengen vom gleichen Typ sind.

In jedem Arbeitszyklus (ein oder mehrere Systemtakte) führt jeder Prozessor seinen ihm eigenen Befehl aus. Die Ein- und Ausgabestellen sind für die einzelnen Prozessoren stets gleich. Deshalb entfällt das Steuerwerk und der Selektionsmechanismus (über Multiplexer) für die Ein- und Ausgabestellen. Jedem dieser Anschlüsse ist ein festes Register zugeordnet. Diese Register sind typischerweise nicht zu Speicherblöcken zusammengefaßt. Den logischen Verbindungen sind Verzögerungswerte zugeordnet, die der Verweildauer der Daten in den Registern entsprechen.

7. SIMD - Rechner.

Eine SIMD Architektur ("single instruction multiple data") besteht in der Regel /SIEGEL-85/ aus einem SIMD Programmspeicher, einem Steuerwerk, einem Hauptspeicher und mehreren gleichartigen Prozessoren. Zu jedem Prozessor ge-

hören mehrere ihm zugeordnete (lokale) Register und ein ausgezeichnetes Datentransferregister (DTR). Für jede ansteigende Flanke von χ_1 wird die Menge der Prozessoren in zwei disjunkte Teilmengen zerlegt. Während die Prozessoren der einen Teilmenge inaktiv sind, führen die restlichen Prozessoren ein und denselben Befehl auf (in der Regel) unterschiedlichen Daten aus. Diese Zerlegung erfolgt mit einer Maskierung. Bis zur nächsten fallenden Flanke von χ_1 hat jeder Prozessor den einen Befehl abgearbeitet und ein event. Ausgabedatum in sein DTR geschrieben. Es sei P die Menge der Prozessoren. Die logischen Verbindungen sind in Form von r Abbildungen $(f_j)_{j=1,\dots,r}$ gegeben mit:

$$\forall j = 1,\dots,r : f_j : P \longrightarrow P \quad \text{mit} \quad \forall p \in P : f_j(p) \neq p$$

Für die logischen Verbindungen W ergibt sich dann:

$$W := \{(p, f_j(p)) : p \in P \text{ und } j = 1,\dots,r\}$$

Damit ist der Ausgang des DTR eines Prozessors p mit den Eingängen der Prozessoren $f_j(p) : j = 1,\dots,r$ verbunden. Der Datenaustausch f_j ist als SIMD Befehl realisiert. Jeder aktive Prozessor übermittelt den Inhalt seines DTR an den Prozessor $f_j(p)$. Dies geschieht in der nächsten 1-Phase von χ_2 .

8. MIMD - Rechner.

Eine MIMD Architektur ("multiple instruction multiple data") besteht aus einem MIMD Programmspeicher, einem Steuerwerk, einem Hauptspeicher und mehreren (unterschiedlichen) Prozessoren. Jeder Prozessor kann über ihm eigene (lokale) Register sowie Datentransferregister verfügen. Ein Steuerwort wird in mehrere Bereiche eingeteilt, wobei ein Bereich jeweils fest einem Prozessor zugeordnet ist. Diese Bereiche stellen die Befehlsleitungen einzelnen Prozessoren dar. Der Befehlssatz eines Prozessors schöpft seine funktionalen Möglichkeiten voll aus, d.h. jede Prozessorkomponente kann angesprochen werden. Die Befehlssätze der einzelnen Prozessoren sind völlig unabhängig von einander. Jeder Befehl b für einen Prozessor p hat eine Ausführungsdauer $\tau_{b,p}$. Wenn ein Prozessor zu $x \in UP(\chi_1)$ die Ausführung des Befehls b beginnt, so müssen zu diesem Zeitpunkt die Eingabedaten von b an seinen Eingängen vorhanden sein. Zum Zeitpunkt $y := \min \{z \in DO(\chi_1) : z \geq x + (\tau_{b,p} - 1) \cdot \beta_{per}\}$ ist die Befehlsausführung beendet und sind event. Ausgabedaten an seinen Ausgängen vorhanden. Da der zu implementierende Al-

gorithmus auf Hauptprogrammebene ein Basisblock ist, liegen schon vor der Ausführung des Programms durch eine Lösung von RCS die Datentransfers fest. Deshalb wird nach Beendigung des Befehls b auf Prozessor p ein Befehl ausgeführt, der die Ausgabedaten in die ihnen zugeordneten Register überträgt.

9. Bereitstellung von Ressourcen.

Die Ressourcen sind Eigenschaften der einzelnen Prozessoren. Es werden zwei Typen von Ressourcen unterschieden:

- Funktionale Eigenschaften :
 - algebraische bzw logische Funktionseinheiten
 - lokale Register
 - Zugriff auf Busse
- Physikalische Eigenschaften:
 - Implementierungsfläche
 - Verlustleistung
 - fan-out.

10. Geschwindigkeit.

Die Ausführungsdauer für Elementaroperationen ist unabhängig von den implementierenden Prozessoren gewählt. Die Geschwindigkeit ist ein Faktor $0 < \delta \leq 1$, um den das Ausführungsintervall einer Elementaroperation verkürzt wird. In der Regel ist jedoch $\delta = 1$.

11. Datenleitungen.

Zwischen den einzelnen Prozessoren bestehen logische Verbindungen. Dadurch entsteht $W \subset v(P)$. Sowohl für Leitungen als auch für Busse sind nur die logischen Verbindungen für die Modellierung von Belang. Falls die Verbindung über Einzelregister erfolgt, kann für alle logischen Verbindungen kann ein fester Verzögerungswert oder eine obere Schranke desselben gewählt werden.

Die Organisation (z.B. Bereichseinteilung, Befehlscodierung) der Steuerwerks geht nicht in die Modellierung ein. Ebenso wird für die Befehle der Prozessoren nur gefordert, daß sie die einzelnen Ressourcen der Prozessoren ansprechen können. Die Realisierung der einzelnen Datentransfers (bit parallel oder bit seriell) spielt für die Modellierung insoweit eine Rolle, als die Länge des Ausführungsintervalls des Zeitbedarf der Datentransfers miteinschließen muß.

Die algorithmische Lösung von GAP erfolgt mit einem modifizierten LTMIS aus /LOUKAKIS-81/. Der Algorithmus ist ein branch and bound Verfahren. Daher wächst die Rechenzeit in Abhängigkeit von $|E|$ sehr schnell, wobei $E \subset T \times P$ die möglichen Zuordnungen darstellt (vergl. (7.3)). Mit Satz (12.17) ist es möglich ein Prototyp RCS zu lösen und daraus eine zulässige Lösung von RCS zu ermitteln. Diese Idee wird nun auf GAP übertragen.

Gegeben sei RCS für ein Programm in $N1$, das eine entfaltete normalisierte Laufschleife enthält.

Es sei $t' : T' \longrightarrow N$ eine zulässige Lösung von P-RCS und sei $t : T \longrightarrow N$ die nach Satz (12.17) entwickelte zulässige Lösung von RCS. Nach (12.17) (ii) gilt:

$$\forall x \in T_{PH} : n \geq \tau(x)$$

Es werden nun zulässige Lösungen $a : E \longrightarrow \{0,1\}$ von GAP gesucht, sodaß:

$$\forall x \in T_{PH} : \exists p \in P : \forall y \in \text{pos}_H^{-1}(x) : a(y,p) = 1$$

Damit werden alle $y \in T_L$ mit $\text{pos}_H(y) = x$ auf demselben Prozessor p strikt sequentiell ausgeführt.

12. Prototyp - GAP

Gegeben sei mit den Bezeichnungen von (7.3) GAP zu einem Prozessornetzwerk (P, W) mit $W = W_f + W_l$, ξ, δ, η .

Definiere nun eine Abbildung $\text{pos} : T \longrightarrow T^*$ mit $T^* := (T \setminus T_L) \cup T_{PH}$

$$\text{pos}(x) := \begin{cases} \text{pos}_H(x) & \text{für } x \in T_L \\ x & \text{sonst} \end{cases}$$

Damit kann das Prototyp-GAP definiert werden:

$$E^* := E \cap (T^* \times P)$$

Gesucht wird eine Abbildung $a^* : E^* \longrightarrow \{0,1\}$, sodaß

$$cTa^* := \sum_{(x,p) \in E^*} c(x,p) \cdot a^*(x,p) \longrightarrow \min$$

unter den Nebenbedingungen:

$$(a) \quad \forall x \in T^* \quad \sum_{p \in N_{E^*}(x)} a^*(x,p) = 1$$

$$(b) \quad \forall p \in P \quad \forall (x,y) \in I_p : \quad a^*(\text{pos}(x),p) + a^*(\text{pos}(y),p) \leq 1$$

$$(c) \quad \forall (x,y) \in R_d$$

$$\forall (p,q) \in v(P) \setminus W : (x,p), (y,q) \in E \quad \Rightarrow \quad a^*(\text{pos}(x),p) + a^*(\text{pos}(y),q) \leq 1$$

$$\forall (p,q) \in W_f : \quad (x,p), (y,q) \in E \wedge \xi(p,q) \neq r_p(x,y) \quad \Rightarrow \quad a^*(\text{pos}(x),p) + a^*(\text{pos}(y),q) \leq 1$$

$$\forall (p,q) \in W_l : \quad (x,p), (y,q) \in E \wedge \xi(p,q) < r_p(x,y) \quad \Rightarrow \quad a^*(\text{pos}(x),p) + a^*(\text{pos}(y),q) \leq 1$$

$$(d) \quad \sum_{p \in \text{prp}(a^* \cdot 1(1))} \eta(p) \leq s$$

P-GAP kann ebenfalls über die Bestimmung von stabilen Mengen im Zuordnungsgraphen gelöst werden.

13. Satz.

Gegeben sei ein Programm in N1, das eine entfaltete normalisierte Laufschleife enthält. Gegeben sei weiterhin RCS für dieses Programm. Es sei t eine zulässige Lösung von RCS, sodaß es eine Lösung a^* von P-GAP gibt. Dann ist die Abbildung

$$a : E \longrightarrow \{0,1\} \quad \text{mit} \quad a(x,p) := a^*(\text{pos}(x),p)$$

eine zulässige Lösung von GAP.

Beweis:

$$\text{Zeige zuerst:} \quad \forall (x,p) \in E \quad : \quad (\text{pos}(x),p) \in E^*$$

Begründung:

Falls $x \in T \setminus T_L$, gilt: $x = \text{pos}(x)$

Falls $x \in T_L$:

Da die entfaltete Laufschleife normalisiert ist, gilt: $\mu(x) = \mu(\text{pos}_H(x)) = \mu(\text{pos}(x))$

Mit Definition (7.3) gilt:

$$(x, p) \in E \Rightarrow \mu(x) \leq \eta(p) \leq s \Rightarrow (*t \text{ zulässig für RCS } *)$$

$$(\mu(\text{pos}(x)) = \mu(x) \leq \eta(p) \leq s) \text{ und } \forall y \in N_R(\text{pos}_H(x)) :$$

$$t(\text{pos}(x)) + \tau(\text{pos}(x)) * \delta(p) \leq t(\text{pos}_H(x)) + \tau(\text{pos}_H(x)) \leq t(\text{pos}_H(x)) + \lambda(\text{pos}_H(x), y) \leq t(y) \Rightarrow (\text{pos}(x), p) \in E^*$$

Zeige nun die Nebenbedingungen (a) bis (d) von GAP:

zu (a):

Sei $x \in T$

Falls $x \in T \setminus T_L$, gilt: $N_E(x) = N_{E^*}(\text{pos}(x))$

$$" \subset " \quad \text{Sei } p \in N_E(x) \Rightarrow (x, p) \in E \Rightarrow (\text{pos}(x), p) \in E^* \Rightarrow p \in N_{E^*}(\text{pos}(x))$$

$$" \supset " \quad p \in N_{E^*}(\text{pos}(x)) \Rightarrow (\text{pos}(x), p) \in E^* \Rightarrow (x, p) \in E \subset E \Rightarrow p \in N_E(x)$$

Falls $x \in T_L$:

" $\subset$ " analog

$$" \supset " \quad p \in N_{E^*}(\text{pos}(x)) \Rightarrow (\text{pos}(x), p) \in E^* \Rightarrow$$

$$\mu(\text{pos}(x)) \leq \eta(p) \leq s \Rightarrow (*\text{Normalisiertheit } *)$$

$$\mu(x) \leq \eta(p) \leq s \Rightarrow (*t \text{ zulässig für RCS } *)$$

$$(x, p) \in E \Rightarrow p \in N_E(x)$$

$$\sum_{p \in N_E(x)} a(x, p) = \sum_{p \in N_E(x)} a^*(\text{pos}(x), p) = \sum_{p \in N_{E^*}(x)} a^*(\text{pos}(x), p) = 1$$

zu (b) und (c): Einsetzen der Definition von a

zu (d): Zeige: $\text{prp}(a^{*-1}(1)) = \text{prp}(a^{-1}(1))$

$$" \subset " \quad p \in \text{prp}(a^{*-1}(1)) \Rightarrow \exists x \in T^* : 1 = a^*(x, p) = (* \text{ Definition von pos } *)$$

$$a^*(\text{pos}(x), p) = a(\text{pos}(x), p) \Rightarrow p \in \text{prp}(a^{-1}(1))$$

" $\supset$ " $p \in \text{prp}(a^{-1}(1)) \Rightarrow \exists x \in T : 1 = a(x, p) = (* \text{ Definition von pos } *)$

$a^*(\text{pos}(x), p) \Rightarrow \exists y \in T^* : 1 = a^*(y, p) \Rightarrow p \in \text{prp}(a^{*-1}(1))$

Damit sind die Mengen, über die summiert wird, gleich. Somit folgt (d). qed

14. Entwurf und Simulation paralleler Rechenanlagen

In diesem Kapitel werden die einzelnen Schritte des Entwurfprozesses beschrieben. Das Ziel besteht darin, aus einem Algorithmus in L unter gewissen Restriktionen ein Modell für einen Parallelrechner zu generieren, auf dem der Algorithmus schnellstmöglich abläuft. Die Beschreibung des Parallelrechners erfolgt zwischen der Registertransfer (RT)- und der Modulebene. Diese Beschreibung enthält folgende Einzelinformationen:

- Anzahl und Typ der benützten Prozessoren (Systemkomponenten).
- Funktionalität der einzelnen Prozessoren (Befehlssatz).
- Anzahl der benötigten Register.
- Verbindungsschema der Prozessoren untereinander und zur Außenwelt.
- Steuerprogramm der einzelnen Prozessoren.

Die Prozessoren entsprechen demnach RT-Modellen von physikalischen Prozessoren.

1. Algorithmus und Elementaroperationen

Der zu implementierende Algorithmus muß mit den sprachlichen Mitteln der Programmiersprache L formulierbar sein. Nun werden im Programmtext Elementaroperationen definiert (12.4). Die funktionale Beschreibung der Elementaroperationen ist der Rumpf der Prozedur, mit deren Aufruf die Elementaroperation identifiziert wird. Falls ein Rechensystem neu entworfen werden soll, orientiert sich die Wahl der Prozessoren eng an den Elementaroperationen. Falls die Implementierung des Programms auf einem bestehenden (auch unvollständig definierten) Rechensystem vorgenommen werden soll, spiegeln die Elementaroperationen die funktionalen Möglichkeiten der verwendeten Prozessoren wieder. Jede Elementaroperation wird als Befehl für einen Prozessor betrachtet. Sie bestimmen damit ganz wesentlich die Komplexität der Einzelkomponenten aus denen das parallele Rechensystem aufgebaut ist.

Falls der Datenpfad eines Mikroprozessors entworfen werden soll, entsprechen Elementaroperationen den Mikrobefehlen (oder auch Folgen davon). Falls Elementaroperationen komplexe Operationen auf integer oder floating point Daten

sind, wird das parallele Rechensystem zumindest systolisch wenn nicht sogar SIMD bzw MIMD sein. Durch die Umwandlung des Programmtextes in N1 (12.8) entstehen u.U. weitere Elementaroperationen. Sowohl die Wahl der Elementaroperationen als auch die der Systemkomponenten ist ein kreativer Vorgang, für den im Rahmen dieser Arbeit noch keine Automatisierung vorgesehen wurde.

2. Kausalitäten und Datentransfers.

Für ein Programm in N1 können nach (12.11) die Kausalitäten R automatisch bestimmt werden. Die Kausalitäten ergeben sich zu diesem Zeitpunkt als reine Datenabhängigkeiten. Entfaltete Laufschleifen (12.13) sollten möglichst normalisiert (12.15) sein, da damit die Wahl eines Prototyps ermöglicht wird. Für die Festlegung der minimalen und maximalen Zeitabstände λ bzw κ auf R spielt der Typ (bzw. die Vorgabe) des parallelen Rechensystems eine wesentliche Rolle. Falls ein systolisches System entworfen werden soll, sind die Verzögerungswerte auf den Datenleitungen zumeist kleiner als 5. Dementsprechend werden für ein $(x,y) \in R$ die Werte $\lambda(x,y) = \tau(x)$ und $\kappa(x,y) = \tau(x) + 5$ gewählt. Für eine MIMD bzw SIMD Architektur ist der ξ - Parameter für Einzelregister zumeist nach oben durch $\xi_{\max}$ beschränkt. Der Wert $\xi_{\max}$ ist die maximale Verweildauer eines Datums in einem Register. Damit werden für ein $(x,y) \in R$ die Werte $\lambda(x,y) = \tau(x)$ und $\kappa(x,y) = \tau(x) + \xi_{\max}$ gewählt. Falls die Implementierung des Programms auf einem vorgegebenen Parallelrechner erzielt werden soll, wird das Minimum $\xi_{\min}$ und das Maximum $\xi_{\max}$ der Verzögerungswerte auf den Datenleitungen berechnet. Für $(x,y) \in R$ werden dann $\lambda(x,y) = \tau(x) + \xi_{\min}$ und $\kappa(x,y) = \tau(x) + \xi_{\max}$ gewählt.

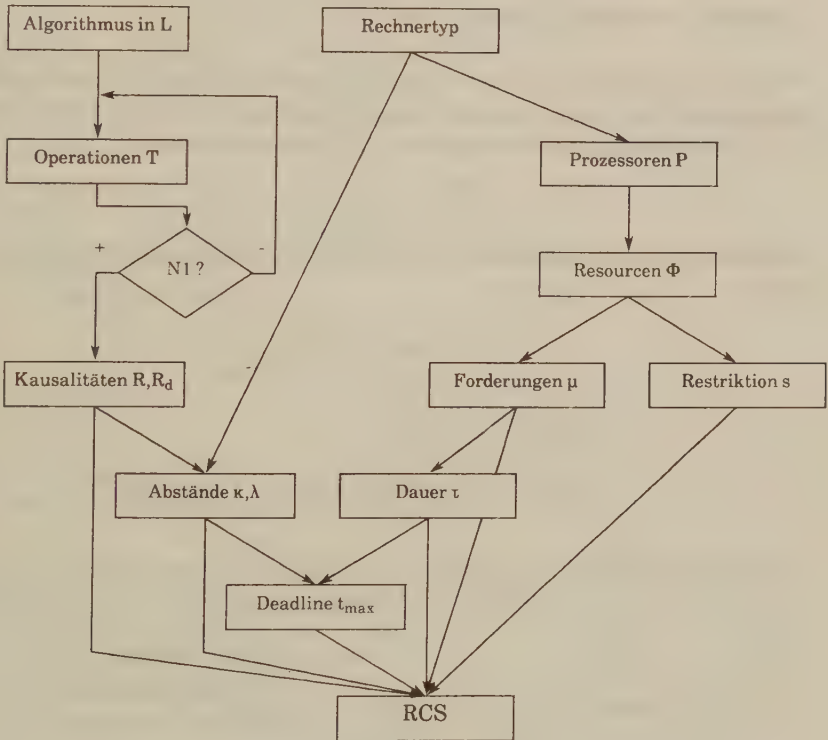
3. Ressourcen und Ausführungsdauer.

Falls ein Programmstück in N1 vorliegt, werden für die Elementaroperationen die Ressourcenanforderungen (12.6) festgelegt. Die Ressourcen zerfallen in zwei Klassen. Die eine Klasse beinhaltet bestimmte Funktionseinheiten (auch Befehle) bzw Systemkomponenten (z.B. Busanschlüsse), die für die Ausführung von Elementaroperationen notwendig sind. Diese Ressourcen werden von den Prozessoren des Systems bereitgestellt. Die zweite Klasse beinhaltet Meßgrößen (Fläche, Verlustleistung, fan-out), die den Prozessoren des Systems zugeordnet sind.

Im Entwurfsfall gibt die Ressourcenbeschränkung Maximalwerte für die einzelnen Ressourcen an. Im Simulationsfall entfällt (in der Regel) die Ressourcenbeschrän-

kung. Die strenge Trennung von Entwurf und Simulation ist jedoch völlig unnatürlich. Oft ist (z.B. durch Handentwurf) eine bestimmte Systemkonfiguration schon grob vorgegeben, jedoch ist noch nach weiteren Details (z.B. welche Leitungen bzw. Prozessoren werden tatsächlich benutzt?) gesucht.

Mit der Festlegung der Ressourcen können die Ressourcenanforderungen (12.6) der Elementaroperationen festgelegt werden. Mit diesen Anforderungen wird dann die Länge des Ausführungsintervalls (12.7) berechnet (geschätzt).



4. Verdrahtung.

Im Entwurfsfall ist die Festlegung der Datenleitungen noch weitgehend offen. Deshalb werden sämtliche logischen Verbindungen zwischen den Prozessoren zugelassen, d.h.

$W := v(P)$, $W_l := W$ und $W_f := \emptyset$.

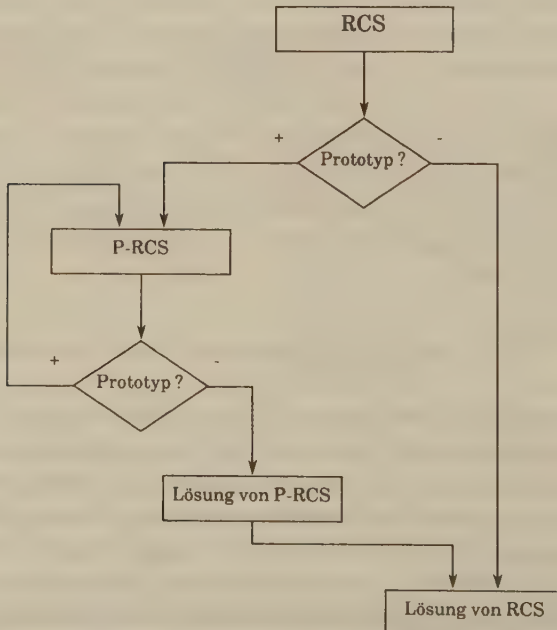
Die obere Schranke für den Verzögerungswert entfällt; d.h. $\xi = \infty$.

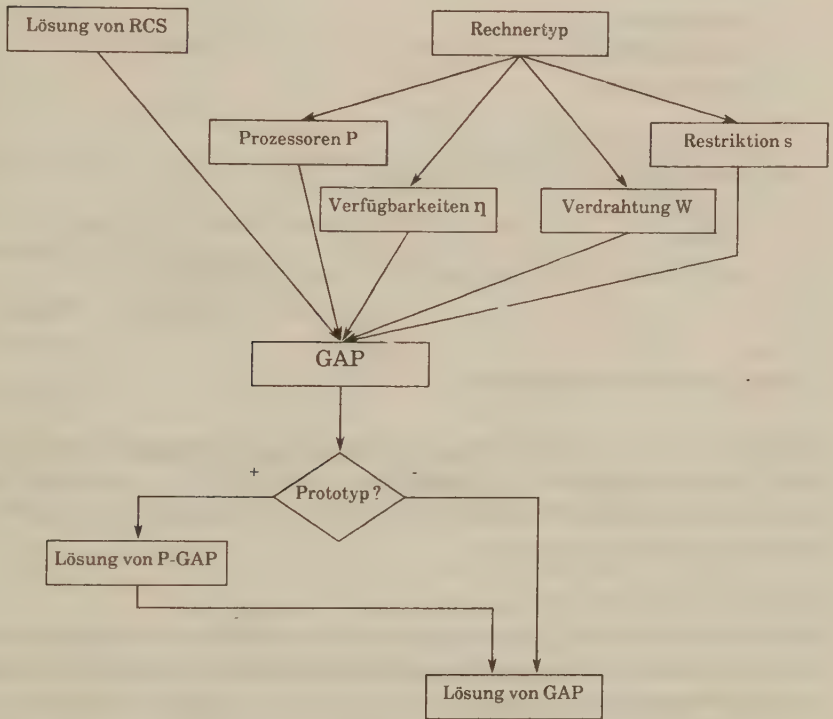
Im Simulationsfall ist W die verfügbare logische Verdrahtung. W_l ist die Menge der Verbindungen mit limitiertem Verzögerungswert wogegen W_f die Menge der Leitungen mit festem Verzögerungswert ist.

5. Spätester Ausführungsbeginn ("deadline")

Die Zeitbeschränkung $t_{\max}$ gibt den spätesten Ausführungsbeginn aller Elementaroperationen an. Dieser Parameter dient dazu, Entwürfe auszuschließen, die eine gewisse Zeitvorgabe überschreiten (z.B. für Echtzeitsysteme).

Mit diesen Festlegungen können nun (P)-RCS und (P)-GAP formuliert und gelöst werden.





6. Schwierigkeiten bei der Lösung von RCS und GAP.

Bei der Formulierung und Lösung von RCS bzw GAP können Schwierigkeiten auftreten. Hier werden Lösungsmöglichkeiten aufgezeigt:

- Zu viele Elementaroperationen:
Lösungsmöglichkeiten:
 - Einführung eines Prototyp.
 - Zusammenfassung mehrerer Elementaroperationen zu einer neuen (z.B. innerste Schleife).
 - Entfernung der äußersten Schleife. Zuerst wird eine Architektur für den Schleifenrumpf entworfen
- Keine Lösung von RCS:
Lösungsmöglichkeiten:

- Erhöhung von $t_{\max}$.
- Erhöhung von s .
- Erhöhung von κ .
- Keine Lösung von GAP:
Lösungsmöglichkeiten:
 - Vermehre W .
 - Erhöhe s .
 - Löse zuerst SAP (11.2) und anschließend das Problem (11.4).

7. Interpretation der Ergebnisse.

Gegeben sei eine zulässige Lösung t von RCS und eine zulässige Lösung a von GAP. Je nach Komplexität der Systemkomponenten entsteht ein Datenpfad für einen Mikroprozessor oder eine MIMD Architektur. Die Menge der benützten Prozessoren ergibt sich aus $\text{prp}(a^{-1}(1))$.

Befehlssatz für die Einzelprozessoren:

Jede Elementaroperation, die auf einem spezifischen Prozessor ausgeführt wird, wird als die Ausführung eines speziellen Befehls des Prozessor interpretiert. Es sei nun p ein Prozessor. Dann ist $T(p) := \{x \in T : a(x, p) = 1\}$ die Menge der Elementaroperationen, die auf p ausgeführt werden. Die Prozedurvereinbarungen der Elementaroperationen aus $T(p)$ stellen den (uncodierten) Befehlssatz für den Prozessor p dar.

Verdrahtung der Prozessoren:

Die logischen Verbindungen, die bei der Abarbeitung des Programms benutzt werden, ergeben sich aus:

$$W := \{(p, q) \in v(P) : \exists (x, y) \in R_d : a(x, p) = 1 \text{ und } a(y, q) = 1\}$$

Die Variablen, die über diese logischen Verbindungen transportiert werden, sind in Einzelregistern (typisch für systolische Systeme) oder in Speicherblöcken (typisch für MIMD Rechner) untergebracht.

Mikro- bzw MIMD-Programm:

Diese Programme sind Folgen von Instruktionen. Die Instruktionen zerfallen in einzelne Bereiche, die jeweils genau einem Prozessor fest zugeordnet sind. Sei $B(p)$ der Befehlssatz des Prozessors p . Es sei $b(a, p)$ der Befehl, der zum Zeitpunkt a dem Prozessor p zugeleitet wird. Dann gilt:

$$b(a,p) := \begin{cases} \text{"Befehl" von } x \in T & \text{falls } \exists x \in T_a : a(x,p) = 1 \\ \text{"idle"} & \text{sonst} \end{cases}$$

Falls ein $x \in T_a : a(x,p) = 1$ existiert, ist es nach (7.4)(b) eindeutig.

Die Instruktion $I(a)$ zum Zeitpunkt a ergibt sich aus:

$$I(a) := (b(a,p))_{p \in P}.$$

Das Mikro- bzw. MIMD-Programm ist dann:

$$(I(a))_{a \in \mathbb{N}}.$$

Spezialfälle:

SIMD Entwürfe entstehen genau dann, wenn gilt:

- $\forall p \in P \quad B(p) = \text{const} \quad (\text{damit hei\ss t der Befehlssatz } B)$
- $\forall a \in \mathbb{N} \quad \exists b_0 \in B \quad \forall p \in P : b(a,p) = b_0 \text{ oder } b(a,p) = \text{"idle"}$

Systolische Entwürfe entstehen genau dann, wenn gilt:

- $\forall p \in P \quad \exists b_0 \in B(p) \quad \forall a \in \mathbb{N} : b(a,p) = b_0 \text{ oder } b(a,p) = \text{"idle"}$

Die Anwendung des Entwurfsvorgangs auf einen Ladder Algorithmus aus der Signalverarbeitung /STROBACH-87/ ist in /GREVEL-87/ nachzulesen. Für die Simulation ist das wichtigste Ergebnis, daß das Verfahren auf dem parallelen Rechen-system implementierbar ist. Neben dem Mikro- bzw MIMD-Programm ist die Gesamtausführungszeit von entscheidender Bedeutung, denn damit kann die Beschleunigung des Programms im Vergleich zu einer strikt sequentiellen Ausführung berechnet werden.

Schlußfolgerung

Die Anwendung von Planungsmethoden des Operation Research zur Synthese von hochspeziellen Computerarchitekturen wurde in diesem Bericht gezeigt. Da der Rechenzeitbedarf zur Lösung von Planungsaufgaben exponentiell mit der Anzahl der zu planenden Objekte wächst, kommt Methoden zur Problemgrößenreduzierung (Kapitel 5 und 12.17) eine entscheidende Bedeutung zu. Die Reduzierung (12.17) war nur durch die Einführung von minimalen und maximalen Zeitabständen möglich. Dies führte zu einer Verallgemeinerung der Problemformulierung für Planungsaufgaben. Die zu dieser neuen Problemstellung bewiesenen Aussagen zielten stets auf algorithmische Lösungsmöglichkeiten bzw. Problemvereinfachungen. Die Formulierung des Anordnungsproblems (GAP) ist völlig neu. Falls der Zuordnungsgraph die Eigenschaft der Perfektheit hat, ist das lineare Anordnungsproblem mit einem kontinuierlichen Ansatz lösbar. Allgemein kann GAP mit einem Algorithmus zur Bestimmung von maximalen stabilen Mengen gelöst werden.

Die vorgestellte Methode kann Grundlage für ein computergestütztes Entwurfssystem von Parallelrechnern sein. Einige Teile eines solchen Systems aus dem Gebiet der Informatik sind in dieser Arbeit ausgespart worden:

- Transformation eines beliebigen L Algorithmus in die Normalform N1.
- Synthaxanalyse von L Programmen.
- Automatisches Ableiten einer Kausalstruktur aus N1 Programmen.

Diese Probleme der Informatik stellen keine prinzipielle Schwierigkeit auf dem Weg zu einer Automation des Entwurfsprozesses dar, aber zu ihrer Realisierung ist ein beträchtlicher Programmieraufwand erforderlich.

Literatur

- /AHO-79/ A. Aho und J.D. Ullman
Principles of Compiler Design
Addison Wesley, Reading Mass., 1979
- /ALLEN-83/ J.R. Allen, K. Kennedy, C. Porterfield, J. Warren
Conversion of controldependence to Data Dependence
ACM Conf. Princ. on Progr. Languages , pp 177 - 189, 1983
- /ARYA-85/ Siamak Arya
An Optimal Instruction-Scheduling Model
for a Class of Vektor Processors
IEEE Trans. on Comp. Vol. C-34, pp 981-995, 1985
- /AYRES-83/ R.F Ayres
VLSI Silicon Compilation and the Art of Automatic Microchip
Design.
Prentice Hall, Eaglewood Cliffs, NJ, 1983
- /CARRE-79/ Bernard Carre
Graphs und Networks
Oxford University Press, 1979
- /CHUANG-84/ H.Y. Chuang und G. He
Design of Problem-Size Independent Systolic Array Systems
Proc. ICCD, pp 152 - 157, 1984
- /CHVATAL-75/ L. Chvatal
On certain Polytopes assoatiated with graphs
J. Comb. Theory B-18, pp 138 - 154, 1975
- /COFFMAN-72/ E.G. Coffman, R.L Graham
Optimal Scheduling for two Processor System
Acta Inform. 1, pp 200 - 213, 1972
- /COFFMAN-76/ Edward Grady Coffman
Computer and Jobshop Scheduling Theory
Wiley, New York, 1976
- /DOLEV-84/ Danny Dolev, Manfred K. Warmuth
Scheduling Precedence Graphs of Bounded Height
Journal of Algorithms 5, pp 48 - 59, 1984
- /DOLEV-85/ D. Dolev
Scheduling Flat Graphs
SIAM J. Comp. 14, pp 638 - 657, 1985
- /DOMSCHKE-81/ Wolfgang Domschke
Logistik I: Transport
Oldenburg, München 1981
- /FISHBURN-85/ Peter C. Fishburn
Interval Orders and Interval Graphs
John Wiley, New York, 1985
- /FORTUNE-78/ S. Fortune, S. Wyllie
Parallelism in Random Access Machines
Proc ACM Symp. Thoery of Comp., pp 114 - 118, 1978

-
- /GARFINKEL-72/ Robert Garfinkel und George L. Nemhauser
Integer Programming
Wiley, New York, 1972
- /GAREY-74/ M.R. Garey und D.S. Johnson
Complexity Results for Multiprocessor Scheduling under
Resource Constraints
Proc. 8th annual Princeton Conf. on Information Science and
Systems, 1974
- /GAREY-75/ M.R. Garey und R.L. Graham
Bounds for the Multiprocessor Scheduling with Resource
Constraints
SIAM J. Comp 4, pp 187 - 200, 1975
- /GAREY-76/ M.R. Garey, R.L. Graham und D.S. Johnson
Resource Constraint Scheduling as Generalized Bin Packing
J. Comb. Theory A21, pp 257 - 298, 1976
- /GOLUMBIC-80/ Martin Charles Golumbic
Algorithmic Graph Theory and Perfect Graphs
Academic Press, New York, 1980
- /GREVEL-87/ M. Grevel
Automatic Synthesis of Algorithm Specified Multiprocessor Systems
wird eingereicht zur Veröffentlichung in IEEE Trans. on Comp., 1987
- /HENNESSY-83/ J. Hennessy und T. Gross
Postpass Code Optimization of Pipeline Constraints
ACM Trans. PLAS 5, pp 422 - 448, 1983
- /HU-61/ T.C. Hu
Parallel Sequencing and Assembly Line Problems
J. ORSA 9 (6), pp 841 - 848, 1961
- /IBARRA-86/ O.H. Ibarra, S.M. Kim, M.A. Palis
Designing Systolic Algorithms Using Sequential Machines
IEEE Trans on Comp. Vol. C-35, pp 531 - 542, 1986
- /JOHNSON-81/ L. Johnson und D. Cohen
A Mathematical Approach to Modelling the Flow of Data
and Control in Computational Networks
in /KUNG-81/ pp 213 - 225
- /KASAHARA-84/ Hironori Kasahara, Seinosuke Narita
Practical Multiprocessor Scheduling Algorithms for Efficient
Parallel Processing
IEEE Trans. on Comp. Vol C-33, pp 1023-1029, 1984
- /KUNG-80/ H.T. Kung, C.E. Leiserson
Systolic Arrays for VLSI
in: Introduction to VLSI Systems, Eds: C. Mead, L. Conway
Addison Wesley, Reading Mass., pp 260 - 292, 1980
- /KUNG-81/ H.T. Kung, R. Sproull und G. Steele (eds)
VLSI Systems and Computations
Computer Science Press, Rockville, Md., 1981
- /LAWLER-76/ E. Lawler
Combinatorial Optimization: Networks and Matroids
Holt, Rinehart and Winston, New York, 1976

-
- /LEISERSON-83/ C.E. Leiserson, J.B. Saxe, F.M. Rose
Optimizing Synchronous Circuitry by Retiming in
Proc. of the 3rd Caltech Conf. on VLSI, ed.: R Bryant
pp 87 - 116, Springer Berlin/Heidelberg, 1983
- /LENSTRA-82/ J.K. Lenstra und A.H.G. Rinnooy Kan (eds)
Deterministic and Stochastic Scheduling
D. Reidel Publ. Co., 1982
- /LI-85/ G.J. Li und B.W. Wah
The Design of Optimal Systolic Arrays
IEEE Trans on Comp. Vol. C-34, pp 66 - 77, 1985
- /LOUKAKIS-81/ E. Loukakis, C. Tsouros
A Depth First Search Algorithm to Generate the Family of Maximal
Independent Sets of a Graph Lexicographically
Computing 27, pp 349 - 366, 1981
- /LUENBERGER-72/ D.G. Luenberger
Introduction to Linear and Nonlinear Programming
Addison Wesley, Reading, Mass., 1972
- /McDOWELL-86/ C.E. Mc Dowell, W.F. Appelbe
Processor Scheduling for Linearly Connected Parallel Processors
IEEE Trans. on Comp. Vol. C-35, pp 632 - 638, 1986
- /MEAD-80/ C. Mead, L. Conway
Introduction to VLSI Systems
Addison Wesley, Reading, Mass., 1980
- /MEHLHORN-84/ Kurt Mehlhorn
Graph Alorithms and NP-Completeness
Springer Berlin Heidelberg, 1984
- /MELHEM-84/ R.G. Melhem und W.C. Rheinboldt
A Mathematical Modell for the Verification of Systolic Networks
SIAM J. Comp. 13, pp 541 - 565, 1984
- /MIRANKER-83/ W.L. Miranker, A. Winkler
Spacetime Representation of Computational Structures
Computing 32, pp 93 - 114, 1983
- /MOLDOVAN-83/ D.I. Moldovan
On the Design of Algorithms for VLSI Systolic Arrays
Proc. IEEE, Vol 71,1, pp 113 - 120, 1983
- /MOLDOVAN-86/ D.I. Moldovan, J.A.B. Fortes
Partitioning and Mapping Algorithms into Fixed Sized Systolic
Arrays.
IEEE Trans. on Comp. Vol. C-35, pp 1 - 12, 1986
- /PAPADIM-79/ Ch. Papadimitriou, M. Yannakakis
Scheduling Interval Ordered Tasks
SIAM J. Comp. 8, pp 405 - 409, 1979
- /QUINTON-84/ P. Quinton
Automatic Synthesis of Systolic Arrays from Uniform Recurrence
Equations
11th Conf. on Comp. Architecture, pp 208 - 214, 1984
- /RAMAKRISH-84/ I.V. Ramakrishnan und P.J. Varman
On Mapping Cupe Graphs onto VLSI Arrays
Proc. Symp. Found. of Softw. Techn. and theor. Comp. Science,
pp 296 - 316, Springer Berlin/Heidelberg, 1984

-
- /SIEGEL-85/ H.J. Siegel
Interconnection Networks for Large Scale Parallel Processing
D.C. Heath & Co., Lexington Mass., 1985
- /SOUTHARD-83/ J.R. Southard
MacPitts: An Approach to Silicon Compilation
IEEE Computer, pp 74 - 82, Dec. 1983
- /STROBACH-87/ P. Strobach
Efficient Covariance Ladder Algorithms For Finite Arithmetic Applications
Signal Processing (North Holland), Juli 1987
- /SUDO-83/ T. Sudo, T. Ohtsuki, S. Goto
CAD Systems for VLSI in Japan
Proc. IEEE Vol. 71 pp 129 - 143, 1983
- /ULLMAN-75/ J.D. Ullman
NP-Complete Scheduling Problems
J. Comp. Systems Science 10, pp 384 - 393, 1975
- /ULLMAN-76/ J.D. Ullman
Complexity of Sequencing Problems
in /COFFMAN-76/ pp 139 - 164
- /ULLMAN-84/ J.D. Ullman
Computational Aspects of VLSI
Comp. Science Press, Rockville Md, 1984
- /UHR-84/ L. Uhr
Algorithm Structured Computer Arrays and Networks
Academic Press, London, 1984
- /WEISER-81/ U. Weiser und A. Davis
A Wavefront Notation Tool for VLSI Array Design
in /KUNG-81/, pp 226 - 234
- /WESTE-85/ N. Weste, K. Eshraghian
Principles of CMOS VLSI Design - A System Perspective
Addison Wesley, Reading, Mass., 1985
- /ZEMAN-83/ J. Zeman und G.S. Moschytz
Systematic Design And Programming of Signal Processors using Project Management Techniques
IEEE, Trans on ASSP, Vol 31, pp 1536 - 1548, 1983
- /ZEMAN-84/ J. Zeman und G.S. Moschytz
A Program Package for the Design, Performance Assessment and Programming of Parallel Signal Processors
IEEE Sympos. CAS, pp 1256 - 1258, 1984

Liste der Problemformulierungen

CLP	clique problem	(9.4)
CLP _{stan}	clique problem in standard form	(9.7)
DM	delay minimization	(10.3)
FP	flow problem	(8.3)
GAP	general allocation problem	(7.4)
GRCS	generalized resource constrained scheduling	(5.1)
LAP	linear allocation problem	(7.5)
LAP _{con}	linear continuous allocation problem	(7.6)
P-GAP	prototype general allocation problem	(13.12)
P-PCS	prototype resource constrained scheduling	(12.16)
RCS	resource constrained scheduling	(4.4)
SAP	simple allocation problem	(11.2)
UTAP	unit task allocation problem	(7.7)
UTS	unit task scheduling	(4.6)

Liste der verwendeten Symbole

Kapitel

a	Lösungsfunktion von GAP	(7)
$\mathbf{a}$	Vektor	(9)
$c(R)$	Komplement der Relation R	(2)
c	Kostenabbildung bzw Pfeilbewertung	(3)
$d(R)$	Dual der Relation R	(2)
due	spätester Ausführungsbeginn einer Elementaroperation	(4)
f	Numerierungsabbildung einer Kette (K, f)	(2)
$h(x)$	Länge der längsten Kette, die in x beginnt	(6)
k	Anzahl der Ressourcenklassen	(4)
$l_c(K, f)$	Länge der Kette (K, f) mit Pfeilbewertung c	(3)
p	Prozessor	(7)
pos	Fortsetzung von pos_H auf ganz T	(13)
pos_H	Position im Hauptteil einer entfalteten Laufschleife	(12)
pos_p	Position im Periodenteil einer entfalteten Laufschleife	(12)
pr_p	Projektion von E auf P	(7)
q	Prozessor	(7)
r	Verzögerungswert	(7)
rel	frühester Ausführungsbeginn einer Elementaroperation	(4)
s	Gesamtverfügbarkeiten von Ressourcen	(4)
$sym(R)$	symmetrische Hülle einer Relation R	(2)
$t(x)$	Anfangszeitpunkt der Elementaroperation x	(4)
$t(R)$	transitive Hülle einer Relation R	(2)
t_{max}	Deadline für den Ausführungsbeginn der Elementaroperationen	(4)
$v(X)$	vollständige Relation über der Menge X	(2)
x	Elementaroperation	(4)
x^*	künstlich hinzugefügte Elementaroperation	(5)
x_L	künstlich hinzugefügte Elementaroperation beim Prototyp	(12)
y	Elementaroperation	(4)
z	Elementaroperation	(4)
A	Teilmenge von Q_R	(5)
C	Clique in (E, Z)	(9)
$\underline{C}$	Menge der maximalen Cliques in (E, Z)	(9)
E	mögliche Zuordnungen von T an P	(7)
G	Graph (gerichtet oder auch ungerichtet)	(2)
$G(A)$	Der von A induzierte Untergraph	(2)
I_p	Inkompatibilitäten	(7)
K	Knotenmenge einer Kette	(2)
M	Binärmatrix	(9)
N	Nachfolgermenge	(2)
P	Menge der Prozessoren	(7)
R	Relation	(2)
R_d	Datentransfers	(4)
Q_R	Quellen einer Relation R (minimale Elemente)	(2)
S	stabile Menge (zumeist in (E, Z))	(10)
S_R	Senken einer Relation R (maximale Elemente)	(2)
T	Menge der Elementaroperationen	(4)
T_L	Elementaroperationen einer entfalteten Laufschleife	(12)
T_{PH}	Elementaroperationen im Hauptteil einer entfalteten Laufschleife	(12)
T_{PP}	Elementarop. im Periodenteil einer entfalteten Laufschleife	(12)
V	Vorgängermenge	(2)
W	Leitungen (Verbindungen) zwischen Prozessoren	(7)
W_l	Leitung mit limitiertem Verzögerungswert	(7)

W_f	Leitung mit festem Verzögerungswert	(7)
X	Grundmenge der Relation	(2)
Z	Kantenmenge über E	(9)
Z	Menge der ganzen Zahlen	(2)
α	Zeitpunkt zwischen 1 und $t_{\max}$	(4)
ξ	Verzögerungswert bzw Verzögerungswertschranke	(7)
δ	Geschwindigkeit der Prozessoren	(7)
ϕ	Resource	(4)
η	Bereitstellung von Ressourcen	(7)
κ	Obergrenze des Zeitabstandes bzw Flußwertes	(4/8)
λ	Untergrenze des Zeitabstandes bzw Flußwertes	(4/8)
μ	Anforderung von Ressourcen	(4)
σ	Stabilitätszahl	(2)
τ	Länge des Ausführungsintervalls	(4)
ω	Cliquenzahl	(2)
ψ	Fluß	(4/8)
Δ	Diagonale	(2)
Φ	Menge der Ressourcen	(4)
$/T/$	Elementzahl der Menge T	(2)
(x,y)	Kante bzw Pfeil eines Graphen	(2)
(K,f)	Kette mit Knotenmenge K und Numerierung f	(2)
$[(x,p),(y,q)]$	Kanten über der Knotenmenge E	(9)
$\lceil \rceil$	kleinste ganze Zahl größer gleich	(6)
$\lfloor \rfloor$	größte ganze Zahl kleiner gleich	(7)

Anhang: Numerisches Beispiel

In diesem Kapitel wird die geschilderte Vorgehensweise auf einen Ladderalgorithmus aus /STROBACH-87/ angewendet. Es wird eine systolische Architektur generiert, die im wesentlichen genauso arbeitet wie die in /STROBACH-87/ vorgestellte Architektur. Der in /STROBACH-87/ vorgestellte Ladderalgorithmus ist eine least square Approximation. Die numerischen Details dieses Verfahrens werden hier nicht angegeben, sondern es wird auf die Darstellung in /STROBACH-87/ verwiesen. Für die hier verfolgten Zwecke genügt es, den Ladderalgorithmus als L Programtext zu betrachten, aus dem eine systolische Architektur abgeleitet werden soll.

Zuerst werden im zugrundeliegende Ladderalgorithmus 'PORLA 1' in Tabelle Nr. 3 aus /STROBACH-87/ die 'stacks' entfernt, was folgende Form des Ladderalgorithmus liefert:

Die Festlegung von $p = 4$ wurde in /STROBACH-87/ vorgenommen. Es folgt nun zuerst der Vereinbarungsteil des Ladderalgorithmus.

Figur 1: Vereinbarungsteil des Ladderalgorithmus in seiner Ausgangsform:

Program ladder_algorithm;

CONST	p	= 4;
	p_minus_1	= 3;
	l_iterat	= 2;
TYPE	index	= -1..p_minus_1;
	time	= 0..l_iterat;
VAR	w,dx,r_out	: ARRAY[index,time] OF REAL;
	x	: ARRAY[time] OF REAL;
	e,c	: ARRAY[index,index] OF REAL;
	ct,r	: ARRAY[index,index,time] OF REAL;
	kf,kb,e_out	: ARRAY[index] OF REAL;
	m,j	: INTEGER;
	t	: time;
PROCEDURE	ww(x,dx,dw	: REAL; VAR w : REAL); EXTERNAL;

Die Prozedur ww wird in der Formulierung des Ladderalgorithmus nicht im detail spezifiziert. Durch die PASCAL Notation der Programmiersprache L ist jedoch das Ein/Ausgaberhalten dieser noch unbekannten Prozedur festgelegt.

Das Hauptprogramm des Ladderalgorithmus wird immer wieder durchlaufen. Deshalb ist die Definition eines Prototyps zwingend, um endlich viele Elementaroperationen zu erhalten. Da in einer Iteration nur Werte, die in dieser oder in der vorangegangenen Iteration berechnet wurden, verwendet werden, ist die Menge $\text{dist}\{T_L\} \setminus \{0\} = \{1\}$ (12.13). Deshalb ist $\text{nkqv} = 1$ (12.14). Deshalb werden genau zwei Iterationen des Ladderalgorithmus betrachtet.

Die Figur 2 zeigt das Hauptprogramm des Ladderalgorithmus.

Figur 2: Hauptprogramm des Ladderalgorithmus in seiner Ausgangsform

```

BEGIN
FOR t := 1 TO 2 DO
BEGIN
ww(x[t],x[t],w[0,t-1],w[0,t]);
dx[0,t] := x[t];
e[0,0] := w[0,t];
r[0,0,t-1] := w[0,t-1];
ct[0,0,t-1] := w[0,t-1];
c[-1,0] := w[0,t];
FOR j := 1 TO p DO
BEGIN
ww(x[t],dx[j-1,t-1],w[j,t-1],w[j,t]);
dx[j,t] := dx[j-1,t-1];
e[j,0] := w[j,t];
c[j-1,0] := w[j,t];
r[j,0,t-1] := w[j,t-1];
ct[j,0,t-1] := w[j,t-1];
END;
kf[1] := -c[0,0] / r[0,0,t-1];
kb[1] := -c[0,0] / e[0,0];
e_out[1] := kf[1] * x[t] + x[t];
r_out[1,t] := x[t] * kb[1] + x[t];
FOR m := 1 TO p-1 DO
BEGIN
e[0,m] := kf[m] * kf[m] * r[0,m-1,t-1] + e[0,m-1] + kf[m] * (c[0,m-1] + c[0,m-1]);
r[0,m,t] := kb[m] * kb[m] * e[0,m-1] + r[0,m-1,t-1] + kb[m] * (c[0,m-1] + c[0,m-1]);
ct[0,m,t] := kf[m] * kb[m] * c[0,m-1] + c[0,m-1] + kf[m] * r[0,m-1,t-1] + kb[m] * e[0,m-1];
c[0,m] := kf[m] * kb[m] * ct[0,m-1,t-1] + c[1,m-1] + kf[m] * r[1,m-1,t-1] + kb[m] * e[1,m-1];
FOR j := 1 TO p-m-1 DO
BEGIN
e[j,m] := kf[m] * kf[m] * r[j,m-1,t-1] + e[j,m-1] + kf[m] * (c[j,m-1] + ct[j-1,m-1,t-1]);
r[j,m,t] := kb[m] * kb[m] * e[j,m-1] + r[j,m-1,t-1] + kb[m] * (c[j,m-1] + ct[j-1,m-1,t-1]);
ct[j,m,t] := kf[m] * kb[m] * c[j,m-1] + ct[j-1,m-1,t-1] + kf[m] * r[j,m-1,t-1] + kb[m] * e[j,m-1];
c[j,m] := kf[m] * kf[m] * ct[j,m-1,t-1] + c[j+1,m-1] + kf[m] * r[j+1,m-1,t-1] + kb[m] * e[j+1,m-1];
END;
kf[m+1] := -c[0,m] / r[0,m,t-1];
kb[m+1] := -c[0,m] / e[0,m];
e_out[m+1] := kf[m+1] * r_out[m,t-1] + e_out[m];
r_out[m+1,t] := e_out[m] * kb[m+1] + r_out[m,t-1];
END;
END;
END.

```

Es werden nun die Elementaroperationen von Hand festgelegt. Die Elementaroperationen beinhalten Programmanweisungen des Hauptprogramms, die auf in etwa gleiche Indizierungen zugreifen. Figur 3 zeigt die Prozedur `set_var`. In ihr werden die Eingabevariable des Ladderalgorithmus bestetzt.

Figur 3: Vereinbarung der Elementaroperation `set_var`:

```
PROCEDURE set_var( xl,dxl,wl : REAL; VAR wo,dxo,e,r,c,ct : REAL);
PROCEDURE ww( x,dx,dw : REAL; VAR w : REAL); EXTERNAL;
BEGIN
ww(xl,dxl,wl,wo);
dxo := dxl;
e := wo;
c := wo;
r := wl;
ct := wl;
END;
```

In der Elementaroperation `reflect_coeff` werden die "Reflexionskoeffizienten" k_f und k_b berechnet. Figur 4 gibt die entsprechende Vereinbarung an:

Figur 4: Vereinbarung der Elementaroperation `reflect_coeff`:

```
PROCEDURE reflect_coeff( e,r,c : REAL; VAR kf,kb : REAL);
BEGIN
kf := -c / r;
kb := -c / e;
END;
```

In der Prozedur `inverse_filter` werden die Ausgabewerte des Ladderalgorithmus berechnet. Die Eingabevariablen des Ladderalgorithmus sind die $x[t]$ $t = 1, 2, \dots$. Die Ausgabewerte sind die $e_out[p]$ und $r_out[p,t]$ $t = 1, 2, \dots$. Auch der Wert $e_out[p]$ wird in jeder Zeiliteration neu berechnet. Sein t index wurde, da immer nur auf Werte einer t -Iteration zugegriffen wird, weggelassen. Figur 5 zeigt die Vereinbarung des Elementaroperation `inverse_filter`:

Figur 5: Vereinbarung der Elementaroperation `inverse_filter`:

```
PROCEDURE inverse_filter( ei,ri,ci,ct : REAL VAR eo,ro : REAL);
BEGIN
eo := kf * ri + ei;
ro := kb * ei + ri;
END;
```

Die eigentlichen Ladderprozeduren werden nun vereinbart. In Figur 6 ist die Vereinbarung von `lad_1_proc` angegeben. Es werden in ihr drei Programmanweisungen zusammengefasst, die in etwa die gleichen Indizierungen haben.

Figur 6: Vereinbarung der Elementaroperation `lad_1_proc`:

```
PROCEDURE lad_1_proc(kf,kb,el,rl,cl,ctl : REAL; VAR eo,ro,cto : REAL);
BEGIN
eo := kf * kf * rl + el + kf * (cl + ctl);
ro := kb * kb * el + rl + kb * (cl + ctl);
cto := kf * kb * cl + ctl + rl * kf + el * kb;
END;
```

In der Elementaroperation `lad_2_proc` bildet die letzte noch unbetrachtete Programmanweisung den Prozedurrumpf. Diese Elementaroperation ist in Figur 7 vereinbart.

Figur 7: Vereinbarung der Elementaroperation `lad_2_proc`:

```
PROCEDURE lad_2_proc(kf,kb,el,rl,cl,ctl : REAL; VAR co : REAL);
BEGIN
co := kf * kb * ctl + cl + kf * rl + kb * el;
END;
```

Figur 8: Ersetzung der Programmanweisungen durch Elementaroperationen

```
BEGIN
FOR t := 1 TO 2 DO
BEGIN
set_var(x[t],x[t],w[0,t-1],w[0,t],dx[0,t],e[0,0],r[0,0,t-1],c[-1,0],ct[0,0,t-1]);
FOR j := 1 TO p DO
set_var(x[t],dx[j-1,t-1],w[j,t-1],w[j,t],dx[j,t],e[j,0],r[j,0,t-1],c[j-1,0],ct[j,0,t-1]);
reflect_coeff(e[0,0],r[0,0,t-1],c[0,0],kf[1],kb[1]);
Inverse_filter(x[t],x[t],kf[1],kb[1],e_out[1],r_out[1,t]);
FOR m := 1 TO p-1 DO
BEGIN
lad_1_proc(kf[m],kb[m],e[0,m-1],r[0,m-1,t-1],c[0,m-1],c[0,m-1],e[0,m],r[0,m,t],ct[0,m,t]);
lad_2_proc(kf[m],kb[m],e[1,m-1],r[1,m-1,t-1],c[1,m-1],ct[0,m-1,t-1],c[0,m]);
FOR j := 1 TO p-m-1 DO
BEGIN
lad_1_proc(kf[m],kb[m],e[j,m-1],r[j,m-1,t-1],c[j,m-1],ct[j-1,m-1,t-1],
e[j,m],r[j,m,t],ct[0,m,t]);
lad_2_proc(kf[m],kb[m],e[j+1,m-1],r[j+1,m-1,t-1],c[j+1,m-1],ct[j,m,t-1],c[j,m]);
END;
reflect_coeff(e[0,m],r[0,m,t-1],c[0,m],kf[m+1],kb[m+1]);
Inverse_filter(e_out[m],r_out[m,t-1],kf[m+1],kb[m+1],e_out[m+1],r_out[m+1,t]);
END;
END;
END.
```

Nachdem in Figur 8 alle Programmanweisungen im Hauptprogramm auf Elementaroperationen verteilt wurden, werden nun die Laufschleifen entfaltet. Dadurch wird der Ladderalgorithmus (Figur 2) in die Normalform N1 transformiert. Dies zeigt Figur 9 und 10:

Figur 9: Das entfaltete Hauptprogramm des Ladderalgorithmus der ersten Iteration ($t = 1$)

```

BEGIN
1  set_var(x[1],x[1],w[0,0],w[0,1],dx[0,1],e[0,0],r[0,0,0],c[-1,0],ct[0,0,0]);
2  set_var(x[1],dx[0,0],w[1,0],w[1,1],dx[1,1],e[1,0],r[1,0,0],c[0,0],ct[1,0,0]);
3  set_var(x[1],dx[1,0],w[2,0],w[2,1],dx[2,1],e[2,0],r[2,0,0],c[1,0],ct[2,0,0]);
4  set_var(x[1],dx[2,0],w[3,0],w[3,1],dx[3,1],e[3,0],r[3,0,0],c[2,0],ct[3,0,0]);
5  set_var(x[1],dx[3,0],w[4,0],w[4,1],dx[4,1],e[4,0],r[4,0,0],c[3,0],ct[4,0,0]);
6  reflect_coeff(e[0,0],r[0,0,0],c[0,0],kf[1],kb[1]);
7  inverse_filter(x[1],x[1],kf[1],kb[1],e_out[1],r_out[1,1]);
8  lad_1_proc(kf[1],kb[1],e[0,0],r[0,0,0],c[0,0],c[0,0],e[0,1],r[0,1,1],ct[0,1,1]);
9  lad_2_proc(kf[1],kb[1],e[1,0],r[1,0,0],c[1,0],c[0,0,0],c[0,1]);
10 lad_1_proc(kf[1],kb[1],e[1,0],r[1,0,0],c[1,0],ct[0,0,0],e[1,1],r[1,1,1],ct[1,1,1]);
11 lad_2_proc(kf[1],kb[1],e[2,0],r[2,0,0],c[2,0],ct[1,0,0],c[1,1]);
12 lad_1_proc(kf[1],kb[1],e[2,0],r[2,0,0],c[2,0],ct[1,0,0],e[2,1],r[2,1,1],ct[2,1,1]);
13 lad_2_proc(kf[1],kb[1],e[3,0],r[3,0,0],c[3,0],ct[2,0,0],c[2,1]);
14 reflect_coeff(e[0,1],r[0,1,0],c[0,1],kf[2],kb[2]);
15 inverse_filter(e_out[1],r_out[1,0],kf[2],kb[2],e_out[2],r_out[2,1]);
16 lad_1_proc(kf[2],kb[2],e[0,1],r[0,1,0],c[0,1],c[0,1],e[0,1],r[0,2,1],ct[0,2,1]);
17 lad_2_proc(kf[2],kb[2],e[1,1],r[1,1,0],c[1,1],ct[0,1,0],c[0,2]);
18 lad_1_proc(kf[2],kb[2],e[1,1],r[1,1,0],c[1,1],ct[0,1,0],e[1,2],r[1,2,1],ct[1,2,1]);
19 lad_2_proc(kf[2],kb[2],e[2,1],r[2,1,0],c[2,1],ct[1,1,0],c[1,2]);
20 reflect_coeff(e[0,2],r[0,2,0],c[0,2],kf[3],kb[3]);
21 inverse_filter(e_out[2],r_out[2,0],kf[3],kb[3],e_out[3],r_out[3,1]);
22 lad_1_proc(kf[3],kb[3],e[0,2],r[0,2,0],c[0,2],c[0,2],e[0,3],r[0,3,1],ct[0,3,1]);
23 lad_2_proc(kf[3],kb[3],e[1,2],r[1,2,0],c[1,2],ct[0,2,0],c[0,3]);
24 reflect_coeff(e[0,3],r[0,3,0],c[0,3],kf[4],kb[4]);
25 inverse_filter(e_out[3],r_out[3,0],kf[4],kb[4],e_out[4],r_out[4,1]);

```

Die Numerierung der Elementaroperationen ist nicht Bestandteil des Hauptprogramms. Mit dieser Numerierung wird auf die einzelnen Elementaroperationen verwiesen.

Figur 10: Das entfaltete Hauptprogramm des Ladderalgorithmus der zweiten Iteration ($i = 2$)

```

26 set_var(x[2],x[2],w[0,1],w[0,2],dx[0,2],e[0,0],r[0,0,1],c[-1,0],ct[0,0,1]);
27 set_var(x[2],dx[0,1],w[1,1],w[1,2],dx[1,2],e[1,0],r[1,0,1],c[0,0],ct[1,0,1]);
28 set_var(x[2],dx[1,1],w[2,1],w[2,2],dx[2,2],e[2,0],r[2,0,1],c[1,0],ct[2,0,1]);
29 set_var(x[2],dx[2,1],w[3,1],w[3,2],dx[3,2],e[3,0],r[3,0,1],c[2,0],ct[3,0,1]);
30 set_var(x[2],dx[3,1],w[4,1],w[4,2],dx[4,2],e[4,0],r[4,0,1],c[3,0],ct[4,0,1]);
31 reflect_coeff(e[0,0],r[0,0,1],c[0,0],kf[1],kb[1]);
32 inverse_filter(x[2],x[2],kf[1],kb[1],e_out[1],r_out[1,2]);
33 lad_1_proc(kf[1],kb[1],e[0,0],r[0,0,1],c[0,0],c[0,0],e[0,1],r[0,1,2],ct[0,1,2]);
34 lad_2_proc(kf[1],kb[1],e[1,0],r[1,0,1],c[1,0],ct[0,0,1],c[0,1]);
35 lad_1_proc(kf[1],kb[1],e[1,0],r[1,0,1],c[1,0],ct[0,0,1],e[1,1],r[1,1,2],ct[1,1,2]);
36 lad_2_proc(kf[1],kb[1],e[2,0],r[2,0,1],c[2,0],ct[1,0,1],c[1,1]);
37 lad_1_proc(kf[1],kb[1],e[2,0],r[2,0,1],c[2,0],ct[1,0,1],e[2,1],r[2,1,2],ct[2,1,2]);
38 lad_2_proc(kf[1],kb[1],e[3,0],r[3,0,1],c[3,0],ct[2,0,1],c[2,1]);
39 reflect_coeff(e[0,1],r[0,1,1],c[0,1],kf[2],kb[2]);
40 inverse_filter(e_out[1],r_out[1,1],kf[2],kb[2],e_out[2],r_out[2,2]);
41 lad_1_proc(kf[2],kb[2],e[0,1],r[0,1,1],c[0,1],c[0,1],e[0,1],r[0,2,2],ct[0,2,2]);
42 lad_2_proc(kf[2],kb[2],e[1,1],r[1,1,1],c[1,1],ct[0,1,1],c[0,2]);
43 lad_1_proc(kf[2],kb[2],e[1,1],r[1,1,1],c[1,1],ct[0,1,1],e[1,2],r[1,2,2],ct[1,2,2]);
44 lad_2_proc(kf[2],kb[2],e[2,1],r[2,1,1],c[2,1],ct[1,1,1],c[1,2]);
45 reflect_coeff(e[0,2],r[0,2,1],c[0,2],kf[3],kb[3]);
46 inverse_filter(e_out[2],r_out[2,1],kf[3],kb[3],e_out[3],r_out[3,2]);
47 lad_1_proc(kf[3],kb[3],e[0,2],r[0,2,1],c[0,2],c[0,2],e[0,3],r[0,3,2],ct[0,3,2]);
48 lad_2_proc(kf[3],kb[3],e[1,2],r[1,2,1],c[1,2],ct[0,2,1],c[0,3]);
49 reflect_coeff(e[0,3],r[0,3,1],c[0,3],kf[4],kb[4]);
50 inverse_filter(e_out[3],r_out[3,1],kf[4],kb[4],e_out[4],r_out[4,2]);
END.

```

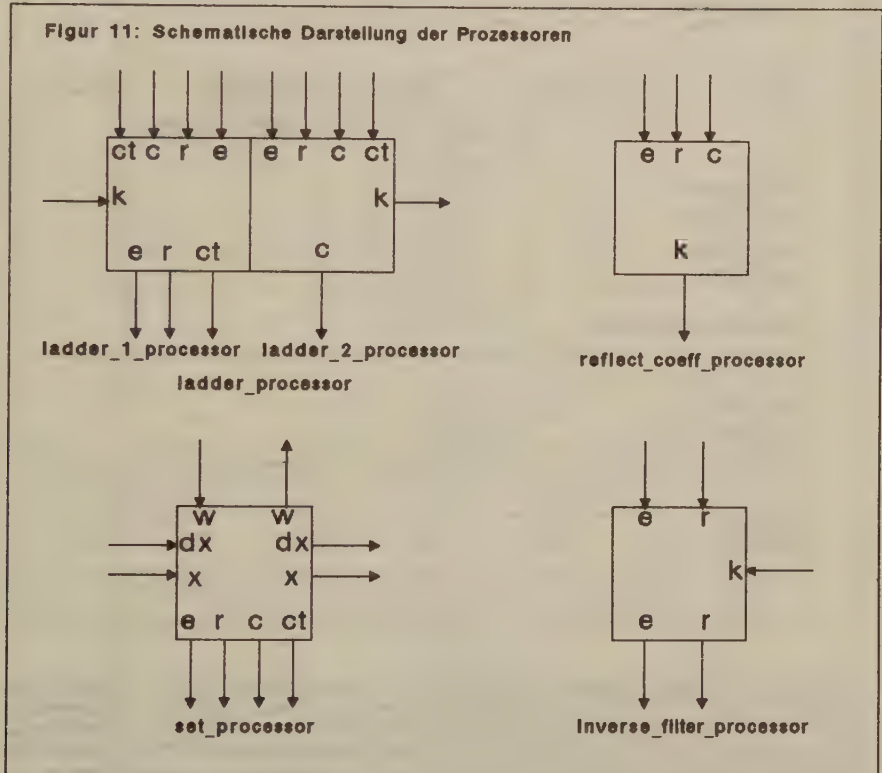
Der Ladderalgorithmus in N1 (12.8) ist der Ausgangspunkt der Formulierung einer Schedulingaufgabe. Die Ausführungsdauer der einzelnen Elementaroperationen ist jeweils gleich eins. Somit kann in einem Systemtakt jede Elementaroperation auf einem entsprechenden Prozessor ausgeführt werden. Die Festlegung der Ausführungsdauer der Elementaroperationen erfolgt in Übereinstimmung mit /STROBACH – 87/.

Die Implementierung der Elementaroperationen erfolgt auf vier verschiedenen Prozessoren:

- **set_processor:** Dieser Prozessor dient zur Implementierung von **set_var**.
- **inverse_filter_processor:** Dieser Prozessor ist eine Implementierung der Prozedur **inverse_filter**.
- **reflect_coeff_processor:** Dieser Prozessor berechnet die 'Reflexionskoeffizienten'.
- **ladder_processor:** Dieser Prozessor kann gleichzeitig eine Elementaroperation **lad_1_proc** und eine Operation **lad_2_proc** ausführen.

In Figur 11 werden nun die verschiedenen Prozessoren schematisch dargestellt. Der ladder_processor hat einen Eingang und einen Ausgang der mit k bezeichnet ist. Ueber diesen Eingang bzw Ausgang werden die Variablen k_f, k_b angeliefert bzw weitergegeben. In ladder_processor erfolgt keine Veraenderung der Variablen k_f, k_b

Figur 11: Schematische Darstellung der Prozessoren



Die funktionalen Eigenschaften der Prozessoren ergeben sich unmittelbar aus den entsprechenden Vereinbarungen (Figur 3 ... 7).

Es werden nun 4 Ressourcen definiert die genau den Prozessoren entsprechen. Jede Elementaroperation fordert damit den Prozessor (die Resource) an, auf dem sie implementierbar ist. Es wird weiters neben Figur 3 ... 7 noch ein weiterer Typ ("additional") definiert, der keine Ressourcen anfordert. Dieser Typ dient dazu die Elementaroperationen 26 ... 50 (Figur 10) darzustellen, denn in diesem Beispiel sollen die Ressourcenbeschränkungen nur fuer die Operationen der ersten Iteration betrachtet werden. Die Details der Eingabeparameter des Schedulingproblems findet sich in Figur 12.

Figur 12: Eingabeparameter der Schedulingaufgabe:

Anzahl Elementaroperationen:	50
Anzahl Ressourcen:	4
letzter Ausfuehrungsbeginn (deadline)	12
Anzahl Typen:	6

Typ	Nummer	Ressourcenanforderung				Dauer
		set	inverse	ladder	reflect	
set var	1	1	0	0	0	1
inverse_filter	2	0	1	0	0	1
lad_1_proc	3	0	0	1	0	1
lad_2_proc	4	0	0	1	0	1
relect_coeff	5	0	0	0	1	1
additional	6	0	0	0	0	1
Gesamtrestriktionen		1	1	2	1	

Elementaroperation	1		5		10		15		20		25												
Typ	1	1	1	1	1	5	2	3	4	3	4	3	4	5	2	3	4	5	2	3	4	5	2

Elementaroperation	26		30		35		40		45		50																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																
Typ	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6

Die Nachfolger bzw. Vorgaengerrelation der 50 Elementaroperationen ist in Figur 13 gegeben. Die zeitliche Verschiebung zwischen zwei aufeinanderfolgende t -Iterationen wird auf eins festgelegt (12.16, 14.2). Damit kann (falls eine Loesung der Schedulingaufgabe existiert) in jedem Systemtakt ein Datum $x[t]$ verarbeitet werden. Da in systolischen Architekturen keine Speicherblöcke vorhanden sind, werden alle logischen Verbindungen als Direktverbindungen realisiert (13.3). Der maximale Verzögerungswert auf einer logischen Verbindung wird auf 5 festgesetzt. Somit gilt fuer die Datenabhaengigkeiten, dass der minimale zeitliche Abstand gleich 1 und der maximale zeitliche Abstand gleich 6 ist. Fuer die durch die Prototypbildung entstandenen Pfeile $(i, i+25)$ $i = 1 \dots 25$ ist der minimale zeitliche Abstand gleich dem maximalen zeitlichen Abstand naemlich 1.

Eine Optimalloesung des so formulierten Schedulingproblems konnte in ca 1000 sec Rechenzeit auf einem MC 68000 System ermittelt werden. Hierbei ist zu beachten, dass diese Loesung schon nach ca 40 sec Rechenzeit errechnet wurde, jedoch wurde die restliche Rechenzeit zum Nachweis der Optimalitaet benoetigt.

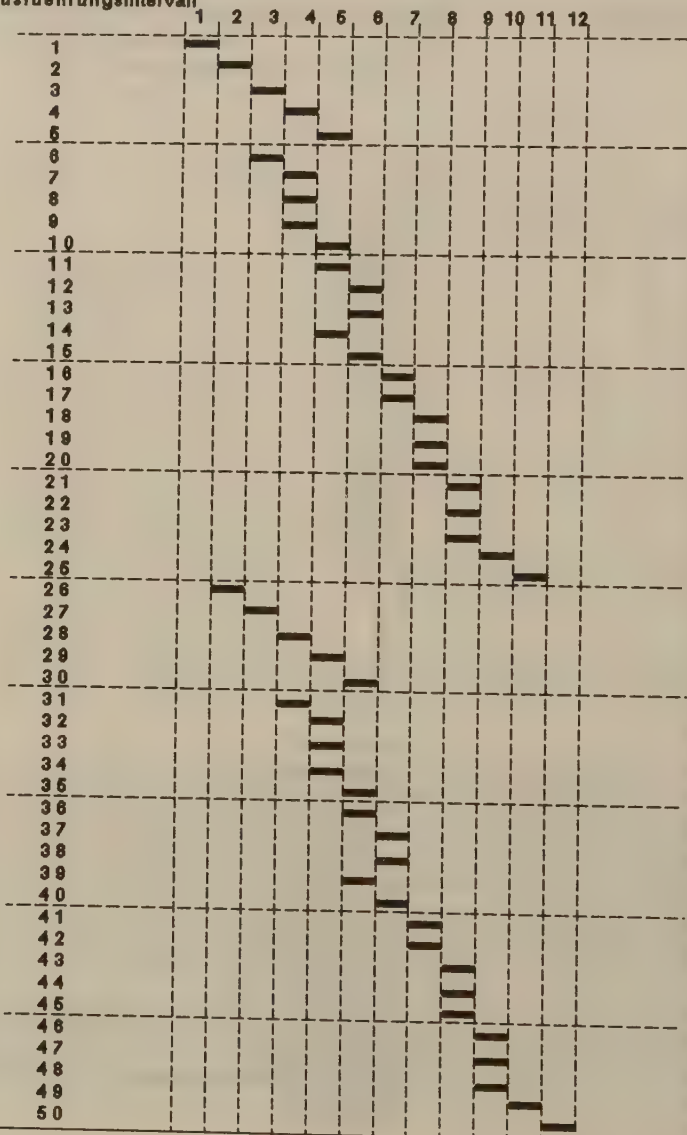
Figur 13: Vorgaengerrelation der Elementaroperationen

Elementaroperation	Vorgaengeroperationen
1	
2	
3	
4	
5	
6	1 2
7	6
8	1 2 6
9	1 2 3 6
10	1 2 3 6
11	2 3 4 6
12	2 3 4 6
13	3 4 5 6
14	8 9
15	7 14
16	8 9 14
17	10 11 14
18	10 11 14
19	12 13 14
20	16 17
21	15 20
22	16 17 20
23	18 19 20
24	22 23
25	21 24
26	1
27	1 2
28	2 3
29	3 4
30	4 5
31	6 26 27
32	7 31
33	8 26 27 31
34	9 26 27 28 31
35	10 26 27 28 31
36	11 27 28 29 31
37	12 27 28 29 31
38	13 28 29 30 31
39	8 14 33 34
40	7 15 32 39
41	8 16 33 34 39
42	8 10 17 35 36 39
43	8 10 18 35 36 39
44	10 12 19 37 38 39
45	16 20 41 42
46	15 21 40 45
47	16 22 41 42 45
48	16 18 23 43 44 45
49	22 24 47 48
50	21 25 46 49

1	
2	
3	
4	
5	
6	1 2
7	6
8	1 2 6
9	1 2 3 6
10	1 2 3 6
11	2 3 4 6
12	2 3 4 6
13	3 4 5 6
14	8 9
15	7 14
16	8 9 14
17	10 11 14
18	10 11 14
19	12 13 14
20	16 17
21	15 20
22	16 17 20
23	18 19 20
24	22 23
25	21 24
26	1
27	1 2
28	2 3
29	3 4
30	4 5
31	6 26 27
32	7 31
33	8 26 27 31
34	9 26 27 28 31
35	10 26 27 28 31
36	11 27 28 29 31
37	12 27 28 29 31
38	13 28 29 30 31
39	8 14 33 34
40	7 15 32 39
41	8 16 33 34 39
42	8 10 17 35 36 39
43	8 10 18 35 36 39
44	10 12 19 37 38 39
45	16 20 41 42
46	15 21 40 45
47	16 22 41 42 45
48	16 18 23 43 44 45
49	22 24 47 48
50	21 25 46 49

Figur 14: Ausführungsintervall der einzelnen Elementaroperationen

Elementaroperation /
Ausführungsintervall



In Figur 14 ist eine Optimalloesung des Schedulingproblems angegeben. Die Ausgangsdaten des Anordnungsproblems werden wie folgt festgelegt:

- Alle Prozessoren haben die gleiche Geschwindigkeit 1.
- Alle logischen Verbindungen zwischen den Prozessoren sind zugelassen.
- Alle logischen Verbindungen haben limitierten Verzoegerungswert mit oberer Schranke gleich 5.
- Die Ressourcenbeschraenkungen werden weggelassen.

Damit die Loesung des Anordnungsproblems fuer beliebig viele t -Iterationen anwendbar wird, wird ein Prototyp GAP (13.12) geloest. Hierbei ist die Menge T^* (aus 13.12) der zuzuteilenden Elementaroperationen gleich der Menge der ersten 25 Elementaroperationen.

Folgende Prozessoren werden in die Architektur aufgenommen:

- 5 Prozessoren vom Typ 'set_processor'
- 4 Prozessoren vom Typ 'reflect_coeff_processor'
- 6 Prozessoren vom Typ 'ladder_1_processor'
- 6 Prozessoren vom Typ 'ladder_2_processor'
- 4 Prozessoren vom Typ 'inverse_filter_processors'

Damit das zu generierende systolische System mit dem in /STROBACH-87/ zu vergleichen ist, werden jeweils ein Prozessor vom Typ 'ladder_1_processor' und einer vom Typ 'ladder_2_processor' zu einem Prozessor vom Typ 'ladder_processor' zusammengefasst. Fuer P-GAP spielen jedoch nur die Prozessoren vom Typ 'ladder_1_processor' $i = 1, 2$ eine Rolle.

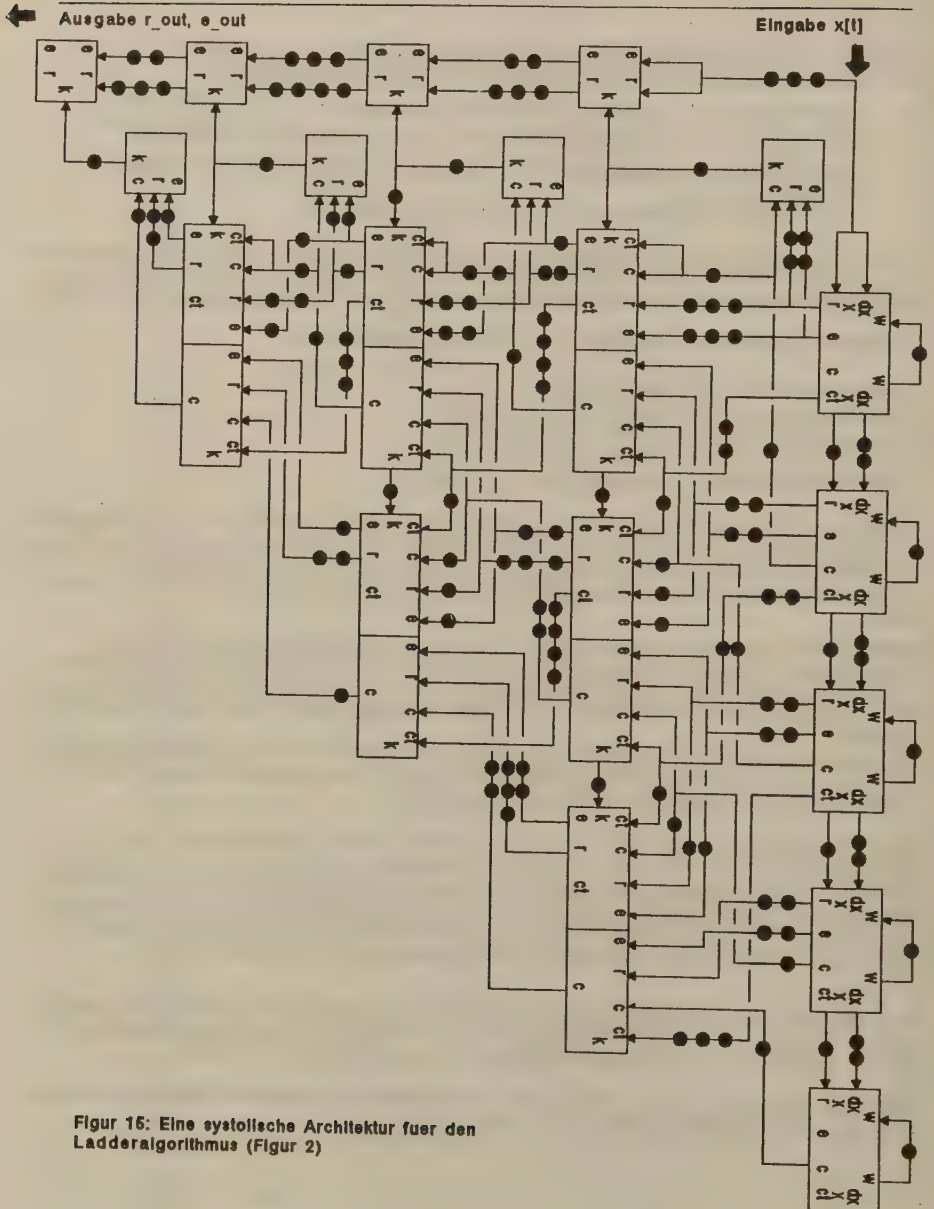
Nach Festlegung der Ressourcen gibt es fuer jede Elementaroperation genau einen Prozessortyp, der sie implementieren kann. Da ich nur an einer zulaessigen Loesung von P-GAP interessiert bin, wird die Kostenfunktion c in (13.12) auf Null gesetzt. Aus einer Prototyploesung wird dann nach (13.13) eine zulaessige Loesung des Anordnungsproblem erzeugt.

Die Loesung des Anordnungsproblems fordert, dass jede der 25 Elementaroperationen auf einen eigenen Prozessor ablaufen muss. Dies ist auch nicht verwunderlich, da der zeitliche Abstand zweier t -Iterationen genau ein Systemtakt ist.

Die 6 Ladderprozessoren sind in der Lage gleichzeitig eine Operation lad_1_proc und lad_2_proc auszufuehren. Die Ressourcenbeschraenkung von set_proc auf 1 verhindert, dass die 5 Setprozessoren fuer eine Iteration gleichzeitig arbeiten. Sonst waere ein Treiber fuer das Eingangssignal x erforderlich, um das Eingangssignal auf alle 5 Prozessoren zu verteilen. Die gleiche Ueberlegung ergibt sich fuer die Beschraenkung der Ladderresource auf 2. Hier muesste das Signal k (k_f, k_b) zumindest verdreifacht werden.

In Figur 15 ist die generierte systolische Architektur angegeben. Die Anzahl der Punkte auf den Verbindungen zwischen Prozessoren gibt die Anzahl der Verzoegerungsregister auf dieser Leitung an. Die Verbindungen arbeiten wie first-in first-out stacks.

Fuer die Architektur in Figur 15 kann nun ein 'retiming' durchgefuehrt werden um die Anzahl der Verzoegerungsregister zu minimieren (10.3).



Figur 16: Eine systolische Architektur fuer den Ladderalgorithmus (Figur 2)

Lebenslauf von Michael Grevel

14. Juni 1957	Geburt in Stuttgart
1964 - 1968	Besuch der Grundschule in Bietigheim (Baden Württ.) und Langau (Niederösterreich)
1969 - 1977	Besuch des Gymnasiums in Horn (Niederösterreich) und Burghausen (Bayern)
24. Juni 1977	Abitur am Aventinus Gymnasium Burghausen
1977 - 1983	Studium der Mathematik (Nebenfach Informatik) an der Ludwig-Maximilians-Universität München
27. Juli 1983	Hauptdiplom in Mathematik
1983 - 1986	Erstellung der vorliegenden Dissertation mit einem Promotionsstipendium der Siemens AG München
1986 - heute	Vorbereitung auf die Doktorprüfung

München, im März 1987

Druck und Verarbeitung: copy shop, darmstadt, tel. 21353